CS 474: Object Oriented Programming Languages and Environments
Fall 2018

First C++ project

Due time: 9:00 pm on Sunday 11/18/2018

You are to implement the back-end of a *Music Library Service* (MLS). The service maintains a set of audio files containing a large number of songs; it will deliver a music file upon demand from a service user.

The back-end of the MLS is not responsible for playing music segments and interfacing to service users; this functionality is implemented in the MLS's front-end, which is not part of your project. However, your back-end will be responsible for maintaining the audio clips, including add and removing clips, editing existing clisps and duplicating audio clips (for the purpose of delivering songs to MLS users).

Each music clip will be implemented by a class called *MusicClip*. The class has the following fields:

1. *ID*—This is a unique (integer) identifier of each song. You are not allowed to use the same identifier for two different songs, even if one of the song has been deleted. Expect that the MLS will store at most one million songs, including deleted songs.

2. *Artist*—This is a string denoting the song's performer which could be a solo artist (e.g., "Katy Perry") or a band (e.g., "Pink Floyd").

3. *Genre*—The is an enumeration with possible values of POP_MUSIC, FOLK_MUSIC, JAZZ, BLUES, CLASSICAL, OPERA, COUNTRY_MUSIC, REGGAE and ROCK.

4. *People*—This is an instance of a separate class named *BandMembers* This is an array of strings giving the names of the artists in the band. This field will be "null" if the song has a solo artist.

5. *Clip*—This is a byte array containing the audio clip. The byte array could be quite large (several megabytes).

6. *Price*—This is the price to be charged customers for the music clip in dollars and cents.

As you develop the MLS's back-end you need not worry filling *MusicClip* instances with actual byte sequences; however, you should make provisions for those byte sequences to be quite large once the MLS is deployed and operational.

Given that each *MusicClip* instance is expected to be quite large, you must store these instances in files most of the time. Each such file will contain information a single instance including its ID, Artist, Genre, etc. Instead you will keep in memory a number of proxies, one proxy for each *MusicClip* instance stored in your file system. Each proxy is an instance of a smart pointer class called *MusicClipPtr* that dynamically reads an instance of class *MusicClip* into memory when the instance is needed and writes it back to file when done.

For the project you must code a test version of the MLS back-end that contains just 10 *MusicClip* instances. Each of the ten instances is stored in a separate file. At any point in time, at most three instances will also reside in memory; the remaining instances will reside only in the ten files.

When the program is started, the program creates a linked list of 10 *MusicClipPtr* instances; however, none of the instances is in memory yet. Next, *MusicClip* instances are read into memory and written

back to files (possibly after being modified) as needed while executing the commands described next. Remember never to hold more than 3 *MusicClip* 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. In addition, you may read commands from an input file. 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 music clips. Each clip is read from its file and its information (ID, Genre, etc.) is displayed in the user's screen. All fields are displayed except for the *clip* field. *MusicClip* instances are written back to files and deleted from memory as needed to keep a maximum of three total memory clips in memory at all times.

2. e—Edit a *MusicClip* instance. This command sets the numbered *MusicClip* to be the current clip. If the clip is not already in memory, it is read from file and assigned to the corresponding *MusicClipPtr* instance. In this case, a *MusicClip* instance currently in memory may have to be written back into its corresponding file and deleted from memory not to exceed the limit of 3 music clips.

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

4. y—Copy the current clip. The current clip is deep copied and stored to a temporary file in the current directory called "temp.clip".

5. i—Read commands from file. This command reads and executes a sequence of commands from a file in the current directory. The file is called "input.inp". All commands in this list can appear in the file, one command per line, except for the i command.

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

7. s—Save all clips. This command save the clip(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 clip's information is saved to the new file. Finally, the stream is closed.

8. q—Quits the MLS.

Implementation notes. Your MLS system must meet the following requirements. First, each *MusicClipPtr* instance should handle exactly one *MusicClip* instance. Second, clients (i.e., your project code) should not be allowed to create or delete *MusicClip* 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 *MusicClip* instance associated with a a given smart pointer instance. Fourth, the API of the *MusicClipPtr* 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 MLS system. The archive should also contain sample text files describing your initial music clips. 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.