Practice Exercise #1 [Weight: ~3% of the Course Grade]

Topic: Object-Oriented Design in C++

- **For this exercise, you must work in pairs. The TAs are available to provide hints.**
- **Include the name and ID for each group member in your file or files.**
- **You do not have to separate class headers and class implementations for this exercise. That is, you may submit your assignment as a single `practice_exercise1.cpp` file.**
- **Please submit your completed assignment before the dropbox closes on LEARN.**
- **For this exercise, do not use pointers or dynamic memory.**

Software Application for Creating Music Playlists

We are designing a software application for creating music playlists.

Step1.

Before starting this step, see Tutorial Notes #0 posted on LEARN for code that shows how to setup classes and implement operator overloading. If not sure where to find the code, speak with the instructor or TAs.

Each piece of music is represented by an instance of `Music` class. For each piece of music, we need to store the artist name, year it was made, and music ID; year it was made is stored as an `unsigned integer` while the other attributes are stored as `string` values.

Implement the corresponding class `Music` that includes the required data attributes, empty constructor, parametric constructor, and overloaded `operator==`. For the empty constructor, store 0 as default year. Also, implement a getter method `string get_artist()` that returns the artist name.

Step2.

Before starting this step, see Tutorial Notes #0 posted on LEARN for code that shows how to setup inheritance. If not sure where to find the code, speak with the instructor or TAs.

A completed piece of music is recorded as an instance of `Song`, which is a derived (child) class of `Music`. For each song, we need to store the genre, song name, and song length; the song length is stored as an `unsigned int` value while others are `strings`.

Implement the corresponding class `Song` that includes the required data attributes, empty constructor, parametric constructor, and overloaded `operator==`. Getters are optional. For the empty constructor, store 0 as default song length.

To use `Song` instance as `Music`, use "`static_cast<Music>(song_info)`".

Step3.

Before starting this step, see Tutorial Notes #0 posted on LEARN for code that shows how to setup and use vectors. If not sure how to interpret the code, speak with the instructor or TAs.

`Playlist` is used to store `Song` instances. Implement the matching class `Playlist`, so that it includes a vector of `Song` instances. Do not implement explicit constructors.

For example, to declare a vector of `Song` instances, write `vector<Song> my_playlist;`

Also, implement methods "`bool insert_song(Song& song_info)`" and "`Playlist shuffle_songs()`".

The `insert_song` method inserts the given song into the `Song` vector; duplicates instances are not allowed, and each playlist must not include more than three songs from the same artist. The `insert_song` method returns `true` if it succeeds in its operation and `false` otherwise.

The `shuffle_songs` method returns a new `Playlist` instance containing the same song instances in the `Song` vector in random order.

HINT: Call "`srand(time(0))`" from `int main()`.

Step4.

Implement an overloaded `operator+` as a non-member `friend` function that combines the two playlists into one and returns a new `Playlist` instance with all the `Songs` included. For this operator, you do not need to check for duplicate songs and/or number of songs from the same artist.

Step5.

Before starting this step, see Tutorial Notes #0 posted on LEARN for explanation of testing concepts. If not sure what is expected, speak with the instructor or TAs.

Write a test (driver) program to test your classes and demonstrate that the specified behaviour was correctly implemented. Include one or more calls for each method specified above including constructors.

Include calls for different variants, such as trying to insert a duplicate song into the playlist, trying to insert four songs from the same artist into the playlist, trying to insert a song after shuffling a playlist, and so on. The driver program should be divided into functions with appropriate names, such as `test_insert_song()` and `test_shuffle_songs()`.

Optionally, you could create separate test classes with test methods as specified above, represent the results of each test using assertions, and then run all the included tests using a method called `run()`.