Practice Exercise #1 Grading Rubric

30 marks in Total

All parts completed according to the given specifications. Subtract marks for each component that is missing or that is not implemented according to specifications. **Provide feedback for subtracted marks.**

Detailed Mark Distribution:

5 marks for the `Music` class implementation (Step1)

- Demonstrated that each artwork is represented as an instance of Artwork class. For each piece of music, artist name, year it was made, and music ID are stored; the year it was made is stored as an unsigned integer while the other attributes are stored as string values. Implemented the empty constructor, parametric constructor, and overloaded operator==. For the empty constructor, 0 is stored as the default year. Implemented getter method `string get_artist()` that returns the artist name.

5 marks for the `Song` class implementation (Step2)

- Demonstrated that once a piece of Music has been completed, it is recorded as an instance of Song, which is a derived (child) class of Music. For each song, genre, song name, and song length are stored; the song length is stored as an unsigned int while others are strings. Implemented the empty constructor, parametric constructor, and overloaded operator==. Getters are optional. For the empty constructor, 0 is stored as the default song length.

11 marks for the `Playlist` class implementation (Step3)

- Demonstrated that `Playlist` is used to store `Song` instances. Implemented the matching class `Playlist` so that it includes a vector of `Song` instances.
- Also, implemented methods "`bool insert_song(Song& song_info)`" and "`Playlist shuffle_songs()`".
- The `insert_song` method implemented so that it 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 implemented so that it returns a new `Playlist` instance containing the same song instances in the `Song` vector in random order.
- "`srand(time(0))`" should be called from `int main()` to avoid resetting the randomization each time the `shuffle_songs` method is called

4 marks for the `Playlist` operator implementation (Step4)

- Implemented overloaded `operator+` function.
- Implemented `operator+` as a non-member `friend` function that combines the two playlists into one and returns a new `Playlist` instance with all the `Song` instances included.

5 marks for the appropriate testing (Step5)

- Wrote a test (driver) program that tests the implemented classes and demonstrates that the specified behaviour was correctly implemented. Implemented one or more calls for each method specified above including constructors.
- Included 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 is divided into functions with appropriate names, such as `test_insert_song() and test_shuffle_songs()`.

If the code does not compile under Dev-C++, try it under another IDE, such as CLion for Windows or XCode for Mac OS. If the compilation issues are minor (e.g., one or two mistakes), correct those. If it still does not compile, make a note of this and award partial grades based on code walkthrough.