Semester Project (Complex Engineering Problem)
CS-212 Object Oriented Programing

As part of OBE paradigm, in this course, you are expected to work on a complex engineering problem as your semester project. This complex engineering problem tests the students for their *Depth of knowledge* (Part-3), selection from *Range of conflicting requirements* (part 2), and *Interdependence* of components (part 4).

Problem statement:

1. "Audio Players" are defined as a media player explicitly designed to play audio files, with limited or no support for video playback. An In-car audio player is an audio player among many other types of audio players. Every audio player has a manufacturer name and a model number. The Incar audio players have a storage media which can be mounted or unmounted. When a media is mounted, player starts playing the first song and then keeps track of current song being played and on unmounting it resets the song tracking. When a song is played, title of the current song is also displayed. Storage media stores a list of songs and it can be of two types namely CD and USB. CD and USB are storage medias, where CD and USB has a fixed storage capacity. A CD can have a limited number of Songs where each song has a unique title and a USB can have a number of Songs with unique titles. The In-car media player only supports CDs of MP3 and USBs up to 16GBs only. If any other CD type or larger USB is mounted the player displays an error message explaining the problem. The In-car media player (discussed in the above scenario) can also be powered on or off. When the player is turned off, its current state (including everything) is persisted in it. And when it is powered back on, that already persisted state is restored.

Now Extend the class ‘Audio Player’ you designed before with an updated datatype ‘Audio Playerv2’ capable to perform following functionalities in addition to the functionalities provided in ‘Audio Player’ class.

- Shuffle playlist – capable to shuffle your current playlist randomly
- Save playlist – capable to save a play list on disk
- Load playlist – capable to load any of the previously created playlists
- Add new audio files to playlist
- Delete files from playlist
- Search an audio file

2. **Design an efficient GUI for your application**

Supplement:

- Help Link for playing audio file in C++
https://www.youtube.com/watch?v=9WeDQHi6sJs

- To access files in directory using C++ you may use following help links
https://codeyarns.github.io/tech/2014-06-06-how-to-use-dirent-h-with-visual-studio.html
https://www.youtube.com/watch?v=Oy97UroDLUA

Assessment Rubrics

Trait	Exceptional [10-8]	Acceptable [7-6]	Amateur [5-3]	Unsatisfactory [2-0]
Application Functionality 15%	Application compiles with no warnings. Robust operation of the application, with good recovery.	Application compiles with few or no warnings. Consideration given to unusual conditions with reasonable recovery	Application compiles and runs without crashing. Some attempt at detecting and correcting errors.	Application does not compile or compiles but crashes. Confusing. Little or no error detection or correction.
Specifications 15%	The program works and meets all of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
Readability 5%	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability 10%	The code could be reused as a whole or each class could be reused.	Most of the code could be reused in other programs.	Some parts of the code require change before they could be reused in other programs.	The code is not organized for reusability.

Object Oriented Design 45%	Well-designed classes with good data abstraction and layering. Complete sets of general operations provided for class interface. Good factoring of common code and use of helper functions. Code is highly maintainable; e.g. a change of representation requires changes in a small number of places. Language features supporting OOP well utilized. A complete encapsulated and abstracted code.	Good data abstraction and some data layering. Good set of operations provided. Code is layered for the most part; little duplication. Many OOP language features used where appropriate.	Some attempt at layering data. Some generalization of interface evident, but missed opportunities. Little attempt at layering code. Code can be maintained with significant effort. Some OOP language features used, but some missed opportunities.	Little attempt at layering data, Missing, duplicate or not useful class interface. Code is a maintenance headache, Violation of encapsulation.
Efficiency 10%	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.