Assignment 4: Playlist COP3330 Fall 2017

Due: Thursday, November 2, 2017 at 11:59 PM

Objective

This assignment will provide experience in working with dynamic arrays of objects as well as working with two classes in a "has-a" relationship. This will also provide some extra practice with array and c-string usage.

ABET / SMALCS Assessment

This assignment is designated as one of the course assignments being used to assess basic programming skills for ABET/SMALCS requirements. Please see the syllabus for details. In addition to the normal grading scales, each student's submission will be judged in several aspects on a scale of Highly Effective, Effective, or Ineffective, as specified by the ABET/SMALCS outcome assessment procedures. A student's submission that earns 70% of the available points will count as an overall score score of Effective.

Task

You will be writing classes that implement a simulation of a playlist for a digital music device. The list will be a dynamic array of `Song` objects each of which stores several pieces of information about a song. You will need to finish the writing of two classes: `Song` and `Playlist`. The full header file for the `Song` class has been provided in a file called `song.h`.

Program Details and Requirements

Using the `Song` class declaration, write the `song.cpp` file and define all of the member functions that are declared in the file `song.h`. Do not change `song.h` in any way, just take the existing class interface and fill in the function definitions. Notice that there are only six categories of songs in this class: `POP`, `ROCK`, `ALTERNATIVE`, `COUNTRY`, `HIPHOP`, and `PARODY`. The expected function behaviors are described in the comments of the header file.

Write a class called `Playlist` with files `playlist.h` and `playlist.cpp`. A `Playlist` object should contain a list of `Songs`. There is no size limit to the song list so it should be implemented with a dynamically allocated array of `Song` objects. You can add any public or private functions into the `Playlist` class that you feel are helpful.

In addition to the `Playlist` class itself, you will also create a menu program (explained below) to manage the playlist. Note that the `Playlist` class should provide most of the functionality as the idea is to build a versatile and reusable class. The menu program you write is just for testing purposes, so the major functionality will be in the `Playlist` class itself. The `Playlist` member functions will be the interface between this menu program and the internally stored data (the list of songs).

Rules for the `Playlist` class:

- All member data of the class `Playlist` must be private
- There should be no cin statements inside the `Playlist` class member functions. To ensure the class is more versatile, any user input described in the menu program below should be done in the menu program itself. Design the `Playlist` class interface so that any items of information from outside the class are received through parameters of the public member functions.
- The list of `Songs` must be implemented with a dynamically allocated array. Between calls to `Playlist` member functions, there should never be more than 5 unused slots in this array (i.e. the number of allocated spaces may be at most 5 larger than the number of slots that are actually filled with real data). This means that you will need to ensure that the array allocation expands or shrinks at appropriate times. Whenever the array is resized, print a message that states that the array is being resized, and what the new size is. Example: "*Array being resized to 10 allocated slots".
- Since dynamic allocation is being used inside the `Playlist` class, an appropriate destructor must be defined to clean up memory. The class must not allow any memory leaks.
- You must use the const qualifier in all appropriate places (const member functions, const parameters, const returns, etc).

Write a main program named `menu.cpp` that creates a single `Playlist` object and then implements a menu interface to allow interaction with the object. Your main program should implement the following menu loop (any single letter options should work on both lower and upper case inputs):

`A`: Add a song to the playlist
`F`: Fine a song on the playlist
`D`: Delete a song from the playlist
`S`: Show the entire playlist
`C`: Category summary
`Z`: Show playlist size
`M`: Show this menu
`X`: Exit the program

Behavior of Menu Selections

Always ask for user input in the order specified. Remember, all user inputs described in the menu objects below should be done by the menu program (not inside the `Playlist` class). Such input can then be sent into the `Playlist` class. The `Playlist` class member function should do most of the actual work since they will have access to the list of songs. For all user inputs, assume the following:

- A song title and artist will always be input as c-string (c-style strings) of maximum lengths 35 and 20 respectively. You may assume that the user will not enter more characters than the stated limits for these inputs.
- When asking for the category, user entry should always be a single character. The correct values are `P`, `R`, `A`, `C`, `H`, and `Y` for Pop, Rock, Alternative, Country, HipHop, and Parody respectively. Uppercase and lowercase inputs should both be accepted. Whenever the user enters any other character for the category, this is an error and you should print an error message and prompt the user to enter the data again (Example error message: "Invalid category. Please re-enter: ").
- User input of the size should be a positive `int` in kilobytes. You may assume that the user will enter an integer. Whenever the user enters a number that is not positive, it is considered an error. When you detect an error, print out an error message and prompt the user to re-enter the size (example: "Must enter a positive size. Please re-enter:").
- User input of menu options are letters. Both upper and lower case entries should be allowed.

A: This menu option should allow the adding of a song to the playlist. The user will need to type in the song's information. Prompt and allow the user to enter the information in the following order: title, artist, category, size. The information should be sent into the `Playlist` object and stored in the list of `Songs`.

F: This option should allow the user to search for a song in the playlist by title or by artist. When this option is selected, ask the user to enter a search string (may assume the user entry will be a string 35 characters or less). If the search string matches a song title, display the information for that song (output format is described in the `operator<<` function that goes with the `Song` class). If the search string matches an artist in the list, display the information for all songs by that artist. If the string matches both a title and an artist (in the same song or across different songs), handle each separately (e.g. handle all title matches and then handle all artist matches). If no matched songs are found in the search, display an appropriate message informing the user that there were no results in the playlist.

D: This option should delete a song from the playlist. When this option is selected, ask the user to type in the title of the song (you may assume that song titles in the list will be unique). Remove this song from the playlist. If there is no such title, inform the user that there were no results in the playlist.

S: This option should print the entire playlist to the screen. Each line should contain one song's information as described in `song.h`. The output should be organized and formatted like a table. That is, you should pad each field so that they line up (either left or right justified per field) across the list of songs. At the end, display the total number of songs in the playlist as well as the total size of the playlist in Megabytes to 1 decimal place.

C: This option should list the playlist contents for one specific category. When this option is selected, ask the user to input a category to print. For the category selected, print out the contents of the playlist (as in the Show option) for only songs matching the selected category. After this, also display the total quantity as well as the total file size in Megabytes to 1 decimal place for all songs in this category.

Z: This option should compute and print the total file storage taken up by the playlist. Display this value in kilobytes.

M: Re-display the menu.

X: Exit the menu program.

General Requirements

- All class member data must be private
- All string usages in this assignment are to be implemented with C-style strings (i.e. null-terminated character arrays). You may not use the C++ string class library. A large point of this assignment is to do your own dynamic memory allocation and management as well as to practice with fixed size C-strings.
- An invalid menu selection should produce an appropriate output message to the user, like "Not a valid option, choose again"
- For all of these options, any output to the screen should be user-friendly. By this, assume that the user of the program has never seen it before. All output should clearly indicate what

information is being presented, and the user should always be told what is expected in terms of input.
- Adhere to the good programming practices discussed in class (no global variable other than constants or enumerations, using `const` in all appropriate places, don't `#include` .cpp files, document your code, etc).
- You may use the following libraies: `iostream`, `iomanip`, `cstring`, `cctype`
- Do not use language or library features that are C++11 only

Extra Credit

Write a function called Sort and add in a menu option using the letter "O" for sorting the playlist. When this menu option is chosen, ask the user whether they want to sort by artist or title ("A" or "T" respectively, allowing for both upper and lower case). Then sort the playlist by ascending lexicographic order on the appropriate field.

Submitting

Archive your `song.cpp`, `playlist.h`, `playlist.cpp`, `menu.cpp`, and `README` files into a simple tar ball (no compression). Submit to the assignment 4 link on blackboard. Make sure the submitted files are named as specified by the syllabus and this writeup, and that you do not include any extra files.

General Advice

- Make sure to double check your blackboard submission to make sure everything works when downloaded.
- Email a copy of your finished homework files to your own FSU account. This email will have a time stamp that shows when they were sent and will also serve as a backup. Useful in case something happens to blackboard.
- Periodically (e.g. nightly) make a backup of your assignment to another machine (e.g. personal computer, linprog, email). Computers die and accidents happen, having a backup prevents you from having to start from scratch. This is also a good feature to have in your makefile as you can abstract the details behind a single call.
- Make sure to include the `README` file as specified in the assignment syllabus http://ww2.cs.fsu.edu/~dennis/teaching/2017_fall_cop3330/docs/syllabus.pdf

COP 3330 Fall 2017 Assignment 4 Due: 2017-11-02 11:59 PM

Student Name: **Grader:**

Grade: **Date Submitted:** **Date Graded:**

Numeric Grade

Description	Earned	Possible	Comments
1. Private Data		5	
2. const Usage		5	
3. C-string Usage		10	
4. Memory Usage		10	
5. I/O Usage		5	
6. Add Song		10	
7. Find Songs		5	
8. Delete Song		5	
9. Show Playlist		10	
10. Show Category		5	
11. Size		5	
12. Menu		3	
13. Exit		2	
12. README		10	
13. Submission		10	

ABET Scores

Description	Effectiveness	Comments
Storage Facilities		
Classes		
Functions		
Control Structures		
Terminal I/O		
Code Quality		
Overall		

Notes: