Assignment 4: Playlist Class

Assigned: October 20, 2016
Due: November 01, 2016, 11:59:59pm

Purpose

The purpose of this assignment is to provide you with experience in working with dynamic arrays of objects. You will also gain more familiarity 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. Note that in addition to the normal grading scales, each student's submission will be judged inseveral aspects on a scale of “Highly Effective”, “Effective”, or “Ineffective”, as specified by ABSET/SMALCS outcome assessment procedures. A student's submission that earns 70% of the available points will count as an overall score of “Effective”.

Project 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.

Documentation

In addition to the project implementation, you will also need to provide a `README` file. Here you will need to document at least 5 errors you encountered and how you solved them. The errors can be some combination of compile errors, linker errors, runtime, and runtime errors. However, you can not use the same errors you used for the first two assignments. The format needs to be a numbered list where each item describes the error type, the error description, and the steps taken to fix the error. For the description, you can just describe the g++ message for compile/linker errors or the crash / wrong output for runtime errors.

You can also include documentation for places where your implementation is unfinished. If you document buggy or unfinished code and describe what the problem is and and general approach to fixing it, then point deductions will be less severe. This shows me that you are aware of problems in your code and that you at least have an idea of how to fix them. This is completely optional, however, it can only help you as I'm not going to change my test code based on your documentation. That is, if you report an error I would have otherwise missed, you won't lose any points for it.

Part 1: Song Class Definition

Using the Song class declaration, write the `song.cpp` file and define all of the member functions that are declared in the `song.h` file. 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 functions behviors are described in the comments of this header file.

Part 2: Playlist Class

Write a class called Playlist with filenames `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 to manage the playlist. Note that the Playlist class should provide most of the functionality since 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.

Rules:

- All member data of 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 that 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 read 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 (for testing purposes) 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, where appropriate).

Part 3: Main Program

Write a main program called `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:   Find 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 programming
```

Always ask for user input in the order specified. Remember, all user inputs described in the menu options 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 functions should do most of the actual work, since they will have access to the list of songs. For all user inputs (keyboard), assume the following:

- A song title and artist will always be input as c-strings of maximum lengths 35 and 20 respectively. You may assume that the user will not enter more characters that 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 and you should print 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, and 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 or artist, display the information for that song (output format is described in the operator<< function that goes with the Song class). If no matching 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 slected, 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 and return to the menu.

S: This option should simply print the entire playlist to the screen, one line per song, in an organized manner (like a table). Each line should contain one song's information, as described in the `song.h` file. Also, display the total number of songs in the playlist as well as the total size of the playlist in Megabytes to 2 decimal places.

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, but for the songs matching the seleced category only (e.g. list all of the Pop songs). After this, also display the total quantity, as well as the total file size (the sum of the sizes), taken up by songs in this category. Print this size in Megabytes to 2 decimal places.

Z: This option should compute and print the total file storage taken up by the playlist, printed out 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 (e.g. no global variables other than constants or enumerations, use const in all approriate places, don't include .cpp files, document your code, etc).
- You may use the following libraries: `iostream, iomanip, cstring, cctype`

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 chose, ask the user whether they want to sort by artist or title (enter 'A' or 'T', allowing upper or lower case). Then, sort the playlist by ascending lexicographic order on the appropriate field (author or title).

Submitting

- Remove all binary files, executables, and driver code
- Use the tar utility to archive your `song.cpp`, `playlist.h`, `playlist.cpp`, `menu.cpp`, and `README` files
- Name the tar file according to the scheme: a4_<*last_name*>_<*first_name*>.tar
- Submit the tar file to the appropriate blackboard assignment link