Programming Assignment #4

Music Player

CS 2308.003 and 004 Fall 2018
Instructor: Jill Seaman

Due: Wednesday, 10/24/2018: upload electronic copy by 9:00am.

Design and implement a Player class that manages the songs on a music player device.

The following class has been provided for you (do not implement this class, the code is available on TRACS/Resources under PA#4 provided code). You should use this class to represent time values in your Player class.

TimeStamp:

The TimeStamp class stores an amount of time in seconds.

These operations are provided for a **TimeStamp**:

two constructors:

- one that takes an integer number of seconds as the argument.
- The other takes one argument, a string containing the time value in this format: "mm:ss" where mm is the minutes and ss is the seconds. The number of minutes must have at least one digit but may have more.

toSeconds() returns the time as a total number of seconds.

toString() returns the time as a string in the "mm:ss" format.

add(string) takes a string in the "mm:ss" format and adds the time represented by that string to the stored time.

subtract(string) takes a string in the "mm:ss" format and subtracts the time represented by that string from the stored time.

bool **greaterThan(string)**: takes a string in the "mm:ss" format and returns true if the stored time is greater (later) than the time represented by the argument string.

Download the files TimeStamp.h and TimeStamp.cpp and incorporate them into your final program. Do not change these files and do not submit them.

Use the TimeStamp class as the data type for the 'length' of the Songs described on the next page, as well as for the total length of all of the Songs stored on the Player.

Player:

You should implement the operations listed below for the Player class. You should use the function descriptions to determine what the member variables should be. However, you must declare a **Song struct** to store the data for a single song (make it a private member), and you must have an **array of Song** structures as a member variable.

Player(string name, float size) a constructor for the class.
'name' is a name for it
'size' is a maximum capacity in Mb.

addSong(string band, string title, string length, float size) a function for adding a new song.
'band' is a name of the band
'title' is a name of the song
'length' is a time length of that song in a '1:23' format ("mm:ss")
'size' is a size of the song in Mb
Return true if the song was added and false if there was not enough space (memory) for it or there was a song with the same band and title already in that device or if the device already has 1000 songs in it.

removeSong(string band, string title) a function for removing a song
'band' is a name of the band
'title' is a name of the song
Return true if the song was successfully removed and false if it wasn't present on the device.

displaySongInfo(string band, string title) a function to display a song's info in the following format:
Band: <the corresponding band's name>
Title: <the corresponding song's name>
Length: <the corresponding song's length>
Size: <a corresponding song's size in Mb>
'band' is a name of the band
'title' is a name of the song
The output must line up in two columns, but the labels may be right justified.

displaySongsByLength() a function to display songs sorted by their time length.

deviceInfo() a function to display info about the device in the following format:
Name: <the player's name>
Number of songs: <the total number of songs>
Total length: <the total time length of all songs>
Free space left: <the amount of Mb of free space left>
The output must line up in two columns, but the labels may be right justified.

Input/Output:

Driver.cpp (provided on TRACS) contains code that you can use to test the Player class (the expected output will be in a comment at the end of the driver file). You do not need to write a main function for the program.

NOTES:

- This program should be done in a Linux or Unix environment, using a command line compiler like g++. Do not use codeblocks, eclipse, or Xcode to compile.
- Take some time to read and understand the instructions AND the provided code before implementing the Player class.
- Use the incremental development strategy! Implement only a few functions at a time (start with the easier ones). Comment the code out in the driver that tests functions you haven't implemented yet.
- Do removeSong last!!
- Create a **makefile** to compile and test the Player class. There will be four goals in this makefile, because you will have three .cpp files. Use the makefile from the timedemo example and modify it to work with these new files. Use the following names for your files:

  TimeStamp.h
  TimeStamp.cpp
  Player.h
  Player.cpp
  Driver.cpp

- Put complete header comments at the top of Player.h and Player.cpp.
- DO NOT change the names of the classes, functions or files.
- You do NOT need to use binary search for this assignment.
- Implement the common part of addSong, removeSong, and displaySong—searching for a given song—in a separate function. Do not duplicate the code for search!
- You may add additional private functions as needed.
- Your program **must compile** and run, otherwise you will receive a score of 0.

Logistics:

Submit 3 files only: Player.h, Player.cpp and makefile

There are two steps to the turn-in process:

1. Submit an electronic copy using the Assignments tool on the TRACS website for this class.

2. Submit a printout of the three files at the beginning of class, the day the assignment is due. Please print your name on the front page, staple if there is more than one page.

See the assignment turn-in policy on the course website (cs.txstate.edu/~js236/cs2308) for more details.