Assignment 3: Playlist Creator

Summary

Many software audio players let the user organize his or her music in various playlists that are saved as separate files without duplicating the music files themselves. In this assignment you will design and develop a program named `jukebox` that implements a simple playlist creator. It will be able to open a database of songs and create a new playlist consisting of a subset of the songs in the database. This project will exercise your knowledge of classes, string processing, and general program organization. It is the largest of the programs you will have written. The program will be run with a single comment line argument that specifies the name of the song database file that the program will open. If there is no command line argument, the program will attempt to open a file in its current working directory named `songs.csv`. If there is no command line argument and no file named `songs.csv` in the current working directory, the program will exit with a useful error message to the user.

Program Description

The program should open a file that contains a database of songs. The input file will contain zero or more songs, one per line. The format of this file is specified below in the section entitled *Input*. If the program is unable to open the song database for one reason or another, it will display an error message and exit. The error handling is described below in the *Input* section. If the program successfully opens the song database file, it will then prompt the user to enter the name of the playlist that he or she wishes to create. Once the user enters the name of the playlist, the program will display a menu of choices to the user. The user will be able to view the songs from the database in various ways and select songs from among the listed songs to add to the playlist. The user will also be able to save the playlist to an output file.

User Interface

The program will display the following menu:

```
[A/a] List all.
[R/r] List all matching an artist keyword ...
[T/t] List all matching a title keyword ...
[V/v] View the playlist.
[S/s] Save
[Q/q] Exit.
```

The details follow.

`List all.` All songs in the database should be printed on the screen , 20 songs at a time, followed by the prompt string "`action:`". The user should either type `next` to see the next 20 songs, or `add` followed by one or more numbers $n_1, n_2, ..., n_k$ to add the song with indices $n_1, n_2, ..., n_k$ to the playlist, or `stop` to stop the display of songs and redisplay the main menu.

`List all by matching an artist keyword ...` The user should be prompted for the keyword to match. A keyword cannot contain blanks or tabs. Then the program should display on the screen, 20 songs at a time, all the songs in the database whose artist matches or contains the keyword, followed by the prompt string "`action:`". The user should either type `next` to see the next 20 songs, or `add` followed by one or more numbers $n_1, n_2, ..., n_k$ to add the song with indices $n_1, n_2, ..., n_k$ to the playlist, or `stop` to stop the display of songs and redisplay the main menu.

`List all by matching a title keyword ...` The user should be prompted for the keyword to match. A keyword cannot contain blanks or tabs. Then the program should display on the screen, 20 songs at a time, all the songs in the database whose title matches or contains the keyword, followed by the prompt string "`action:`". The user should either type `next` to see the next 20 songs, or `add` followed by one or more numbers $n_1, n_2, ..., n_k$ to add the song with indices $n_1, n_2, ..., n_k$ to the playlist, or `stop` to stop the display of songs and redisplay the main menu.

`View the playlist.` This is how the user can delete songs from the playlist. All songs in the playlist should be printed on the screen , 20 songs at a time, followed by the prompt string "`action:`". The user should either type `next` to see the next 20 songs, or `remove num` to remove the song with index `num` from the playlist.

`Save.` The playlist should be saved to a file whose name matches the name of the playlist. (See the format of the output file below). If the file exists, it will be overwritten by the new playlist[1].

Exit. The program exits.

Displaying a list of songs. When the user chooses any of the first four menu items above, the program should display the list of songs, 20 songs at a time (a smaller number of songs should only be printed on the last screen). Each song should be displayed on a single line using no more than 80 characters (since this is the typical width of the terminal window). You should organize the information as follows:

```
NNNN. AAAAAAAAAAAAAAAAAAAA TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT BBBBBBBBBB MM:SS YYYY
```

`NNNN.` is a four digit, right justified index number of a song in the playlist followed by a single dot.

`AAAAAAAAAAAAAAAAAAAA` is a twenty character wide, left justified name of the artist, possibly with spaces.

`TTTTTTTTTTTTTTTTTTTTTTTTTTTTTT` is a thirty character wide, left justified title, possibly with spaces.

`BBBBBBBBBB` is ten character wide, left justified name of an album containing the song.

`MM:SS` is the duration of the song displayed using exactly two digits for minutes, followed by the colon, followed by exactly two digits for seconds.

`YYYY` is the year associated with the song.

If the exact artist name, title, or album name does not fit in those limits, they should be truncated. If it is too short, the rest of the allotted characters should be filled with spaces.

Each entry on the line is separated by a single space.

A sample display may appear as follows:

```
   1. Adele                Melt My Heart To Stone         19         03:23 2008
   2. Adele                First Love                     19         03:10 2008
  13. Norah Jones          Wish I Could                   Not Too La 04:18 2007
  44. Norah Jones          Not Too Late                   Not Too La 03:31 2007
 137. Joni Mitchell        One Week Last Summer           Shine      04:59 2007
 139. Joni Mitchell        This Place                     Shine      03:54 2007
 231. Sara Bareilles       Love Song                      Little Voi 04:20 2007
5202. Sia                  Little Black Sandals           Some Peopl 04:14 2007
```

Note that the index numbers may not be consecutive if the search results are displayed.

A typical terminal is 24 rows tall. If 20 songs are displayed followed by 3 blank lines, the prompt will appear on bottom line. If the user has resized the terminal so that it is shorter than 24 rows, the program will not display nicely. Your program does not have to contend with this problem.

After the user enters an action in response to the prompt, if the program again displays 20 songs followed by 3 blank lines, the prompt will again be on the bottom line. This is how you can control to a limited extent how "nice" your program's display looks.

[1] In C++ using the iostream library, it is not easy to alter this behavior. It is much easier when using the C library's file I/O functions to prevent a file from being overwritten.

Input

As noted above, the input file is either a command line argument or the default `songs.csv` file in the current working directory. Whichever it is, when the program tries to open it, if the open fails, it must exit and report this error on the standard error stream.

If the file is opened successfully, it must be read into an appropriate data structure in the program's memory.

The input file is a tab-separated text file. This means that each field on a line is separated from the adjacent fields by a single tab character. In addition, each field is delimited by double quotes.

Each line, except for the first one, should contain a description of a single song. The first row in the file will contain a set of headings. The headings are always the same in a valid input file:

```
"Name"  "Artist"    "Album" "Genre" "Size"  "Time"  "Year"  "Comments"
```

If the heading in the input file does not match the above ones, the program should detect such an input file as invalid and report on the standard error stream that the input file heading is not valid.

The remaining lines contain descriptions of songs. Each line contains eight fields, corresponding to the above named headings. Although your program must display the time using the format `mm:ss`, the time stored in a file is given in seconds. An example might look like:

```
"Melt My Heart To Stone" "Adele" "19" "R&B" "6772679" "203" "2008" "my favorite"
"First Love" "Adele" "19" "R&B" "6651186" "190" "2008" "it's ok"
"Wish I Could" "Norah Jones" "Not Too Late" "Jazz" "8536495" "258" "2007" "from Otis"
"Not Too Late" "Norah Jones" "Not Too Late" "Jazz" "6913447" "211" "2007" "from Otis"
"One Week Last Summer" "Joni Mitchell" "Shine" "Pop" "9936667" "299" "2007" ""
"This Place" "Joni Mitchell" "Shine" "Pop" "7695743" "234" "2007" ""
"Love Song" "Sara Bareilles" "Little Voice" "Pop" "4210858" "260" "2007" "send to Joe"
"Little Black Sandals" "Sia" "Some People Have Real Problems" "Pop" "" "254" "2007" ""
```

Only the first two fields: *Name* and *Artist* are mandatory on the line. Any field past the first two can be left blank. A blank field is indicated by an empty set of quotes (not even white spaces should be allowed). For example, if the Album, Genre, Size and Comments are all blank, an entry could look as follows (with tabs separating the entries):

```
"One Week Last Summer"  "Joni Mitchell" ""  ""  ""  "299"   "2007"  ""
```

Validation of the input file. Summarizing the validity of an input file, a valid file must satisfy all of the following conditions:

- the headings must be valid and all present;
- the first two fields on each line must be non empty;
- each line has to contain exactly eight fields;
- the fields are separated by tabs (and therefore there are seven tabs in each line).

No other checks on the validity must be performed. You can assume that the following will be true in every input file that is valid according to the above conditions:

- the time is always specified as the number of seconds;
- the size is the number of bytes;
- there are no empty lines, but each line ends with a newline character (specifically, the last line in the file will have a newline at the end).

Output

The format of the output file should be identical to the format of the input file. It should contain the same heading and each line should be in exactly the same format as an input line's format. This way, an output file from one run of the program can be used as an input file for another run of the program.

Programming Rules

Your program must conform to the programming rules described in the Programming Rules document on the course website. Remember in particular that:

- You must write your program yourself. I will on occasion ask you to explain the code to me, so you need to make sure that you understand the code that you write. If you want to use features of C++ that we did not cover in class, you need to understand them thoroughly.
- You must document your program (preamble, function pre- and post-conditions, comments throughout the code, appropriately choosing variable and function names).
- All input and output must be performed by the `main` function only, i.e. no other functions should perform any I/O operations. The functions that perform the computational tasks must pass all data through their parameter lists.
- You must use a class to represent a single song.
- You must use a class to represent a playlist.
- Late programs lose 20% for each 24 hours they are late.

Grading

The program will be graded based on the following rubric.

- Does the program compile (on a cslab machine): 10%
- Correctness and implementation of the song class: 10%
- Correctness and implementation of the playlist class: 20%
- Correctness and implementation of the user interface: 15%
- Correctness and implementation of the file I/O operations: 10%
- Correctness and implementation of `main()` and any other features in the program: 15%
- Correctness of multiple file implementation and modularity: 10%
- Documentation: 10%

Submitting the Assignment

This assignment is due by the end of the day (i.e. 11:59PM, EST) on December 10, 2012. You need to submit multiple files for this assignment, so they must be placed into a directory and zipped up as the instructions below describe. The name of your directory must be `*username*_hwk3.` where `*username*` is to be replaced by your username on our system. Do not name it anything else. If it is named anything else, you lose 3% of the program's value. Before you submit the assignment, make sure that it compiles and runs correctly on one of the cslab machines. Do not enhance your program beyond this specification. Do not make it do anything except what is written above.

You are to create the above named directory and place all of your source code files into it. Do not place any executable files or object files into this directory. You will lose 1% for each file that does not belong there. With all files in your directory, run the command

```
zip -r username_hwk3.zip ./username_hwk3
```

This will compress all of your files into the file named `username_hwk3.zip`. Put this zip file into the directory

```
/data/biocs/b/student.accounts/cs135_sw/cs135projects/project3
```

Give it permission 600 so that only you have access to it. To do this, `cd` to the above directory and run the command

```
chmod 600 username_hwk3.zip
```

If you put a file there and then decide to change it before the deadline, just replace it by the new version. Once the deadline has passed, you cannot do this. I will grade whatever version is there at the end of the day on the due date.