Playlist manager

Objectives

1. Practice dynamic memory allocation
2. Practice composition of objects
3. Practice operator overloading

Project details

You will write a PlayList class that manages a play list of songs. The PlayList class manages a dynamically allocated array of song objects. The song class is provided for you in song.h and song.cpp, which should NOT be modified. Your playlist class will be implemented in two files proj5\_playlist.h and proj5\_playlist.cpp. The class will support two basic functions:

1. Managing the playlist. This includes adding and deleting songs.
2. Playing from the play list. This includes keeping track of which is the next song to be played. Obviously we will not actually be playing any music. “Playing” the song will consist of printing the name.

To support these functions, your PlayList class should have the following public member functions:

* The class should have a default constructor that builds an empty playlist with space for 2 songs.
* **void AddSong(const Song &s);**  
  This function will add s to the end of the play list. This function needs to adjust the song array size when necessary. This function should not affect the current song index (for playing songs).
* **bool DeleteSong(const Song &s);** This function tries to delete s from the play list. If s is in the list, the function deletes the song from the list and returns true. Otherwise, nothing is done and the function returns false. If the play list has multiple copies of one song, you should just delete one copy. This functions needs to adjust the song array size when necessary to make sure that the number of songs in the list is at least half the size of the capacity. If after a song is deleted, the playlist object becomes empty, the function must make the object identical to the one created by a default constructor. This function should reset the current song index only if the index is out of bound after a song is deleted.
* **void ShowAll() const;**  
  This function will print all of the songs in the play list on the screen.
* **void play(int num=1);** This function “plays” num songs from the list. The songs in the list should be play in a circular manner: when reaching the end of the list, repeat from the beginning.
* **void ShowStatus() const;** This function shows the status information of the playlist including the total slots allocated for the song list (size of the song array), the number of songs in the list, and the current song index.
* The class should have a destructor that deletes the memory allocated for the playlist when the PlayList object is deallocated.

The PlayList class must also overload the following operators and implement the described functionality accordingly:

1. **Addition operator (+)**: adding a song to a playlist (i.e. playlist object + song object)  
   This routine will return a new playlist object. The returned playlist will be a concatenation of all songs in the caller’s playlist and the single song at the end (same as the result of AddSong).
2. **Subtraction operator (-)**: removing a song from a playlist (i.e. playlist object – song object).This function will return a new playlist object that has the same content as the result of DeleteSong.

All data members in this class should be private. To support the public and friend functions, the PlayList must maintain a number of data members. The most important one is the playlist that stores an array of songs. However, since there is **no size limit** on the play list, it must be declared as a Song pointer (e.g. Song \*plist;) so as to be used as a dynamically allocated array. The dynamic array must expand and shrink to ensure that its size is no more than twice that number of songs in the list or no more than 2. More specifically, there are three situations:

* At initialization, the size of the dynamic array should be 2.
* In the AddSong function, if the dynamic array is full (array capacity = number of songs), the array size needs to double to make room for new songs.
* In the DeleteSong function, if after a song is deleted, the number of songs is less than ½ of the array capacity, the array size needs to be halved (with the exception that the capacity is 2 as in the initialized empty playlist, in which case, the array should not shrink).

You should isolate two memory management functions in two routines: *void doublesize()* and *void halfsize().* The taskes in both functions are similar with the following steps:

1. Allocate a new dynamic song array of the desired size, (2) copy the content in the old song array to the new array, (3) free the old array (plist), (4) set plist to the new array and set bookkeeping variables accordingly. For diagnostic purposes, you must call ShowStatus() at the beginning of each function and before each function returns (see the behavior in the sample executable). Doublesize will be called in the AddSong function while halfsize will be called in the DeleteSong function.

The class must also contain other private member for bookkeeping with at least three private data members: an int member to record the capacity of the dynamic array, an int member to record the number of songs in the dynamic array, and an int member to record the current song index (to support the play function). These members are displayed in the ShowStatus function.

Test Program

A test program menu.cpp is provided. This program will allow you to interactively test the features of your PlayList class. Keep in mind that this program does not test everything that your class must handle. For example, it doesn't test the operator overloads. You should write other tests to verify you implement your memory allocation and deallocation correctly. To compile the program: put all files in one directory and issue ‘g++ song.cpp proj5\_playlist.cpp menu.cpp’

Submission

The due time for this assignment is June 26 (Wendesday), 2013. 11:59pm.

Tar all of your files for this assignment, which should include two files proj5\_playlist.cpp, and proj5\_playlist.h, name the tar file yourlastname\_firstinitial\_proj5.tar and submit the tar file in blackboard.

Grading policy

The program must work on linprog. O point for programs with any g++ compiler error on linprog.

* ‘g++ song.cpp proj5\_playlist.cpp menu.cpp’ has no compiler error (20 points)
* Default constructor and ShowStatus work correctly (30 points)
* AddSong with correct dynamic array resizing, including diagnostic messages (15 points)
* ShowAll (10 points)
* Play (10 points)
* DeleteSong with correct array resizing including diagnostic messages (15 points)

Hints

1. Start the project as soon as possible. This project is the hardest so far.
2. You will need to decide your private data members first, once that do that you should proj5\_playlist.h file that contains the playlist class declaration.
3. You should then create a shell implementation of the class in proj5\_playlist.cpp that can be compiled with song.cpp and menu.cpp (20 points).
4. After that, you should implement the default constructor and the ShowStatus function, both are trivial (30 points)
5. After that you can work on the doublesize function (which has the four steps described earlier) and the AddSong function.
6. After that you can work on the ShowAll and Play functions, and then halfsize and DeleteSong.