**MCS 253P**

**HW 1**

I recommend using C++, but Java is fine too. If you aren't fluent in C++ and plan to use it, first complete this C++ [tutorial](https://www.cprogramming.com/tutorial/c++-tutorial.html). It doesn’t take long. If you only know Python and don’t think you can learn either Java or C++ in time, please ask me and I may allow you to use Python for this course. Most companies want you to know all three of these languages well and if you only stick to the one you know well, you may not every learn and master the others.

Also, first complete [homework 0](https://docs.google.com/document/d/1hadmThgYa0MiOHSF58vY8MQYm7Boowj7ftx84LGiihc/edit?usp=sharing) which ensures that you have your account set up on open lab and are able to use all the tools that will be using this quarter. If you prefer to use Java, substitute the appropriate Java commands for C++ commands.

The purpose of this homework is to get you used to breaking up your program into smaller functions. We want to give each function a good name, which is descriptive of what it does, and good parameters which also have names which indicate what they are. Please do not get intimidated by this programming assignment. Just handle it one function at a time and you will get it done before you know it.

[Read this first on what to submit for a homework.](https://docs.google.com/document/d/1Gfy08swH0b0HSJ4kXGJLBvraCGz8ywxvmzDQhTQv3QM/edit?usp=sharing)

Write a C++ or Java, called MusicLibrary, that allows the user to manage their music collection. Each MusicLibrary has a name specified as an argument to the command line. If none is given, use the default name of myMusic. When the program is run, the named MusicLibrary is loaded from the file (name of file matches name of MusicLibrary) into memory. If the file does not exist, the in-memory MusicLibrary will be empty. When the program is exited, the MusicLibrary is saved back to the file from which it came. Put your entire program in one file named MusicLibrary.c. You will compile and run your program with one of the following commands

g++ MusicLibrary.cpp -o MusicLibrary

MusicLibrary

javac MusicLibrary.java

java MusicLibrary

A MusicLibrary is a list of songs - each of which has a title, artist, and year published. Note we may add more fields in the future, but for now, just store those three values. Songs are retrieved or deleted by using their title. You may limit each title and artist to 40 characters.

Your main program will be a loop that prints a prompt “myMuisc Command: “ where myMusic will be the name of the current MusicLibrary. Each command is a single letter and either upper or lower case letters are treated the same. Any invalid command character is skipped and ignored.

Implement the following additional commands:

I - insert a new song, prompt for title, artist, and year published, maintain the list in ascending alphabetical order by song title, ignore letter case when doing comparisons, but retain letter case in all stored information

P - print out all the entries in this MusicLibrary

D - delete a specified song, prompt for the title of the song to be deleted, delete all matching songs

L - lookup the song for a specified title, prompt for the title to look-up, use [binary search](http://en.wikipedia.org/wiki/Binary_search_algorithm) which only works on sorted list.

Q - save the current MusicLibrary to its file and exit (quit) the program.

Here is a sample session starting with myMusic. “**bash $**” is the Linux command shell prompt, so I am running the program in the first command and after the last Q, my program has exited and I am back at the command prompt where I give a command to [cat](http://en.wikipedia.org/wiki/Cat_(Unix)) myMusic so you can see the three entries as they are stored in the file myMusic. Note, you may store the entries in the file in your choice of format.

**bash $** MusicLibrary

i

**Title:** C my title

**Artist:** C my artist

**Year Published:** 2014

i

**Title:** A my title

**Artist:** A my artist

**Year Published:** 2014

i

**Title:** B my title

**Artist:** B my artist

**Year Published:** 2014

p

**1 Title: A my title, Artist: A my artist, Year Published: 2014**

**2 Title: B my title, Artist: B my artist, Year Published: 2014**

**3 Title: C my title, Artist: C my artist, Year Published: 2014**

L

**Title:** B my title

**Title: B my title, Artist: B my artist, Year Published: 2014**

D

**Title:** B my title

P

**1 Title: A my title, Artist: A my artist, Year Published: 2014**

**2 Title: C my title, Artist: C my artist, Year Published: 2014**

Q

**bash $** cat myMusic

**Title: A my title, Artist: A my artist, Year Published: 2014**

**Title: C my title, Artist: C my artist, Year Published: 2014**

**Design Sketch** This is how I structured my program and you must follow my design for your program (to help you get used to identifying useful functions as you write programs). You must keep your functions reasonably small so your program is understandable - no function body over 5 lines, except switch statement. Large functions quickly become difficult to read, understand, and debug. Lines with a single curly brace do not count towards the maximum of 5 lines nor does the function prototype count. Variable declarations do not count either and you may declare multiple variables of the same type in one line, e.g., int i = 9, j = 0;

Remember that C++ is a one-pass language, so items must be declared BEFORE they are referenced (called). If you use Java, put your entire program in one main class. It often works best to write programs upside-down with main function at the bottom of the file, then above it are the functions main calls, then above them are the functions they call, and so on. Otherwise, you will have to repeat the function declaration above their first use. Having the same information in two places makes it difficult to change things while you program.

Write your entire program in one file. At the top of your file, declare the structure for a Song which contains data members for each item you plan to store in a MusicLibrary entry.

struct Song {

string title;

string artist;

int year\_published;

};

Do not use dynamic allocation for this program. That means you will never call **new** or **delete**. Use a fixed size global array of Songs for your MusicLibrary. You should pick an upper bound on the size (1024 is fine) and declare that maximum size as a symbolic constant defined with #define, then you must declare your array of Songs of the appropriate size. You must keep a count of how many songs are currently in the MusicLibrary so you know how many are in the array and where the next available array slot is located.

#define SONGMAX 1024

struct Song music\_library[SONGMAX];

int current\_number\_of\_songs = 0;

You must include iostream and fstream so you can do file I/O. [here's an example.](http://www.cplusplus.com/doc/tutorial/files/) If you call exit(), you must include stdlib.h. You may want to include string.h so you can use strcmp() to compare two C strings. [Example of strcmp](http://www.tutorialspoint.com/c_standard_library/c_function_strcmp.htm) You will use strcpy() to assign characters from one char array to another.

Start with a **main** procedure (at the bottom of the file). It can determine the name of the MusicLibrary file (defaulting to “myMusic”), then load the saved data into the memory structure, Then it can loop reading a user command then evaluating that command. When that terminates, it can save the MusicLibrary back to the file. You may declare your song array local to main if you prefer, but you must pass the array into each function that is called. For this program, it is fine to declare the song array as a top-level (global) variable.

If you have experience with object-oriented programming languages, think of this program as a Singleton class that has data members for the array and the count of songs and the functions you write are methods on the class.

**read\_command** can print a prompt, then read a character from the user using getchar(). Skip any whitespace characters (space, tab, newline).

**evaluate\_command** can take a command character and decide which command it is, then do the appropriate action.

**load\_MusicLibrary** and **store\_MusicLibrary** load or store a named MusicLibrary file into the memory MusicLibrary. Refer here for an example of reading/writing from/to files [Link](http://www.cplusplus.com/doc/tutorial/files/)

I wrote a command **print\_MusicLibrary** that may be used for both storing the MusicLibrary to a file or for printing the MusicLibrary on the terminal for the P command. I passed in a boolean argument to indicate if I want line numbers (for terminal print) or not (for file print).

**crunch\_up\_from\_index(i) and crunch\_down\_from\_index(i)** Are auxiliary functions that each use a for loop to copy items up or down within the array from a given index. They will be used to insert or remove an item from the music list. They are expensive can you compute the time complexity of each one If N is the number of songs in the song list?

**find\_index\_of\_song\_with\_name** returns the location of the song with the specified name (used by remove and lookup). This is the function that should use binary search and return the index of where it found the song or where the song should be if it were in this array list.

**remove\_song\_from\_MusicLibrary\_by\_name** removes a song with the specified name. You will find it with int i = find\_index\_of\_song\_with\_name(), then crunch\_up\_from\_index(i).

**add\_song\_to\_MusicLibrary** takes a song and puts it in the MusicLibrary in memory in the proper location. You could use i = find\_index\_of\_song\_with\_name(...), then crunch\_down\_from\_index(i), then buf[i] = song.

**write\_song** and **read\_song** handle writing and reading songs from a specified file.

**open\_file** takes a specified file name and a file mode, either “r” or “w” and opens that file, does error checking, then returns the file handle (FILE \*).