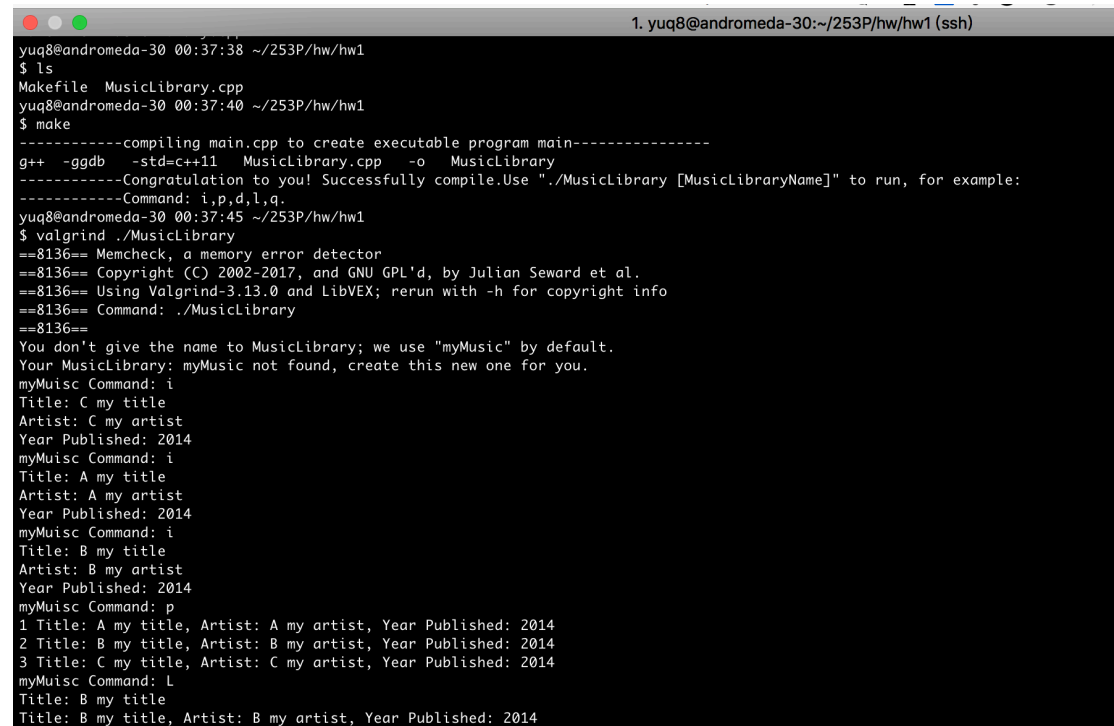

Name: Yu Qin

Results Screenshots

=====start of the write-up=====

Test command:



```
1. yuq8@andromeda-30:~/253P/hw/hw1 (ssh)
yuq8@andromeda-30 00:37:38 ~/253P/hw/hw1
$ ls
Makefile  MusicLibrary.cpp
yuq8@andromeda-30 00:37:40 ~/253P/hw/hw1
$ make
-----compiling main.cpp to create executable program main-----
g++ -ggdb -std=c++11 MusicLibrary.cpp -o MusicLibrary
-----Congratulation to you! Successfully compile.Use "./MusicLibrary [MusicLibraryName]" to run, for example:
-----Command: i,p,d,l,q.
yuq8@andromeda-30 00:37:45 ~/253P/hw/hw1
$ valgrind ./MusicLibrary
==8136== Memcheck, a memory error detector
==8136== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==8136== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8136== Command: ./MusicLibrary
==8136==
You don't give the name to MusicLibrary; we use "myMusic" by default.
Your MusicLibrary: myMusic not found, create this new one for you.
myMusic Command: i
Title: C my title
Artist: C my artist
Year Published: 2014
myMusic Command: i
Title: A my title
Artist: A my artist
Year Published: 2014
myMusic Command: i
Title: B my title
Artist: B my artist
Year Published: 2014
myMusic Command: 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
myMusic Command: l
Title: B my title
Title: B my title, Artist: B my artist, Year Published: 2014
```

```

myMusic Command: L
Title: B my title
Title: B my title, Artist: B my artist, Year Published: 2014
myMusic Command: D
Title: B my title
myMusic Command: 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
myMusic Command: q
==8136==
==8136== HEAP SUMMARY:
==8136==      in use at exit: 73,272 bytes in 2 blocks
==8136==    total heap usage: 3 allocs, 1 frees, 73,840 bytes allocated
==8136==
==8136== LEAK SUMMARY:
==8136==    definitely lost: 0 bytes in 0 blocks
==8136==    indirectly lost: 0 bytes in 0 blocks
==8136==    possibly lost: 0 bytes in 0 blocks
==8136==    still reachable: 73,272 bytes in 2 blocks
==8136==          suppressed: 0 bytes in 0 blocks
==8136== Rerun with --leak-check=full to see details of leaked memory
==8136==
==8136== For counts of detected and suppressed errors, rerun with: -v
==8136== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
yuq8@andromeda-30 00:39:25 ~/253P/hw/hw1
$ cat myMusic
Title: A my title, Artist: A my artist, Year Published: 2014
Title: C my title, Artist: C my artist, Year Published: 2014
yuq8@andromeda-30 00:39:32 ~/253P/hw/hw1
$ █

```

Question & solution: `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? $O(n)$

Conclusion: right result, 0 error in memory leak.

Hw1 main design framework:

- 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.
- 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.

Actually, we used to include "MusicLibrary.h".

But since "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.

- main program will be a loop that prints a prompt "myMusic Command: ". 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.
- You must include iostream and fstream so you can do file I/O. [here's an example](#). 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](#) You will use strcpy() to assign characters from one char array to another.
- 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.

Comments & future suggestions:

- This homework is a well-designed project. The struct design, function split and process, etc., are good practice.
- In later/real-world development, this music library can include the feature of importing from other format of music library, means the feature of sorting at first importing or transition between similar keyword, etc., which will require another elaborate processing.