**MCS 253**

**HW 4**

[Read this first on what to submit for a homework.](https://drive.google.com/open?id=1vvTZzNrXNrxFCFVuzryEuAX6c8rfGERsEc7t4TS1vAA)

***HINT:*** If you write good hierarchical object-oriented code, the assignment will become a lot easier!!!

**HW 4.1 (50 points)**

A company you are working for is asking you to write a program, ***musicPlayer***, that creates a playlist of songs (and their artist) in a specific order. They want you to implement your own simple data type called ***SimplePlayList*** to act as the playlist. Each song’s title and the artist should be stored together, but as separate attributes. Because space is very expensive for the company (given the high number of users they support), they want ***SimplePlayList*** to be minimalistic in terms of space (ie. it should only contain attributes that are needed). An empty playlist should use minimal memory. Your main program, ***musicPlayer***, should utilize a ***SimplePlayList*** object and be able to process the following commands by the user:

* ***push:*** adds a song to the front of the playlist in *O(1)* time.
* ***queue:*** adds a song to the end of the playlist in *O(n)* time.
* ***current:*** displays the current song, its previous song, and its next song in *O(n)* time.
* ***delete:*** deletes current song in *O(1)* time. Current moves to the next song.
* ***prev:*** makes the previous song the new current song in *O(n)* time (the “*prev”* of the first song should be the last song).
* ***next:*** makes the next song the new current song in *O(1)* time (the “*next”* of the last song should be the first song).
* ***restart:*** makes the first song in the list the new current song in *O(1)* time
* ***find***: in *O(n)* time finds the queried song, and in *O(1)* time from when the song is found, is able to display it, its previous song*,* and its next song.
* ***changeTo:*** changes current song to the song entered by the user in *O(n)* time
* ***addBefore:*** adds a song before another existing song in *O(n)* time
* ***addAfter:*** adds a song after another existing song in *O(n)* time
* ***random:*** makes a random song the new current song in *O(n)* time
* ***print***: displays the playlist (in order) in *O(n)* time

Input:

* the above mentioned commands from stdin

Output:

* the above mentioned outputs to stdout

Example:

input:

push Mundian To Bach Ke [Panjabi MC]

push My Immortal [Evanescence]

queue California Love [Tupac]

next

current

addBefore Mundian To Bach Ke [Panjabi MC] Canta Per Me [Yuki Kajiura]

addAfter Mundian To Bach Ke [Panjabi MC] Shape Of My Heart [Sting]

current

prev

prev

delete

changeTo California Love [Tupac]

current

find Mundian To Bach Ke [Panjabi MC]

print

output:

the current song is: California Love [Tupac]

the previous song is: Mundian To Bach Ke [Panjabi MC]

the next song is: My Immortal [Evanescence]

the current song is: California Love [Tupac]

the previous song is: Shape Of My Heart [Sting]

the next song is: My Immortal [Evanescence]

the current song is: California Love [Tupac]

the previous song is: Shape Of My Heart [Sting]

the next song is: My Immortal [Evanescence]

cannot find song!

the playlist is:

1. My Immortal [Evanescence]

2. Canta Per Me [Yuki Kajiura]

3. Shape Of My Heart [Sting]

4. California Love [Tupac]

Make sure your code is robust! The test input will check for it!

**Instead of showing three edge cases in your report, answer the following questions** (no code please):

1. What data structure did you implement ***SimplePlayList*** as?
2. List all the attributes (aka fields) you needed in order to implement ***SimplePlayList*** (also include the attributes for any other auxiliary data structures it uses)? (Do not list functions (aka methods)).
3. How does ***SimplePlayList*** retrieve a random song in *O(n)* time? Explain in detail using a few sentences.
4. If ***prev*** is processed in *O(n)* time, then how is ***find*** able to print the previous song of the found song in *O(1)* time?

**HW 4.2 (25 points)**

Your company decides they can afford a little more space usage and want to improve some of the program’s features. Create a new class, ***GeneralPlayList***, that is based on ***SimplePlayList***, with the same interface but using slightly more space to achieve improved complexities. The following ***musicPlayer*** commands are improved as indicated:

* ***queue:*** adds a song to the end of the playlist in ***O(1)* time**.
* ***current:*** displays the current song, its previous song, and its next song in ***O(1)* time**.
* ***prev:*** makes the previous song the new current song in ***O(1)* time** (the “*prev”* of the first song should be the last song).
* ***random:*** makes a random song the new current song (**never iterating over a song twice**)

As before, do not use any more space than needed to achieve this goal.

Example:

(See 4.1)

Make sure your code is robust! The test input will check for it!

**Instead of showing three edge cases in your report, answer the following questions** (no code please):

1. Very briefly explain how ***GeneralPlayList*** differs from ***SimplePlayList*** as? (Just one or two short sentences that gets the main point(s) across).
2. List all the attributes (aka fields) you needed in order to implement ***GeneralPlayList*** (also include the attributes for any other auxiliary data structures it uses)? (Do not list functions (aka methods)).
3. What allows ***queue*** to be able to add to the end of the playlist in *O(1)* time now?
4. How does ***GeneralPlayList*** retrieve a random song without iterating over a song twice? Explain in detail using a few sentences.

**HW 4.3 (25 points)**

Space is no longer a concern for your company and they will allow you to use existing standard library data structures. They now want you to augment the existing implementation of ***GeneralPlayList*** to improve upon it. Create a new class, ***AdvancedPlayList***, that is based on ***GeneralPlayList***, such that the following ***musicPlayer*** commands are further improved as indicated:

* ***changeTo:*** changes current song to the song entered by the user in ***O(1)* time**
* ***addBefore:*** adds a song before another existing song in ***O(1)* time**
* ***addAfter:*** adds a song after another existing song in ***O(1)* time**

Example:

(See 4.1)

Make sure your code is robust! The test input will check for it!

**Instead of showing three edge cases in your report, answer the following questions** (no code please):

1. Very briefly explain how ***AdvancedPlayList*** differs from ***GeneralPlayList*** as? (Just one or two short sentences that gets the main point(s) across).
2. What additional data structure did you use to help you achieve the desired improvements?

**Possible LeetCode Substitutions (25 points each)**

**Important Notes:**

* You may only substitute 50pts-worth of LeetCode problems

Possible LeetCode Problems:







