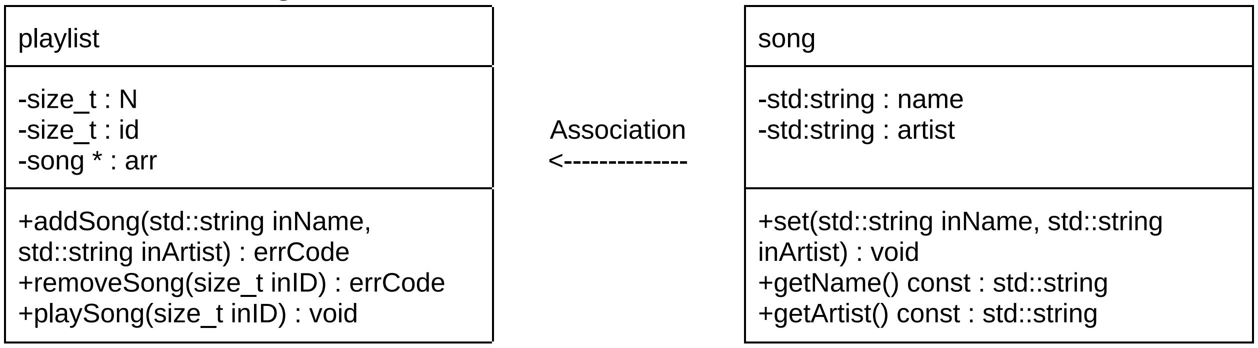
1. **Overview of Classes**
   - What classes did you design?
     - I designed a **song** and **playlist** class
   - Describe the role of each class in your design?
     - The **song** class has data members **name** and **artist** that holds songs
     - The **playlist** class has a dynamically allocated array of songs called **arr**
   - How do the classes relate to one another?
     - The **playlist** class has an array of **songs**

2. **UML Class Diagram**


playlist
-size_t : N
-size_t : id
-song * : arr
+addSong(std::string inName, std::string inArtist) : errCode
+removeSong(size_t inID) : errCode
+playSong(size_t inID) : void
Association
<--------------
song
-std:string : name
-std:string : artist
+set(std::string inName, std::string inArtist) : void
+getName() const : std::string
+getArtist() const : std::string

Note: errCode is just: typedef bool errCode;

3. **Details on Class Decisions**
   - Song class
     - Data members
       - **std::string Name** - stores the name of the song
       - **std::string Artist** - stores the name of the artist
     - It actually doesn't need a constructor or destructor because it only has **string** data members
       - This means when song goes out of scope, the string destructors will be called
       - I added constructors and destructors to set the data members to an empty string for pragmatic reasons

   - Playlist class
     - Data members
       - **size_t N** - holds the size of arr, is size_t
       - **size_t id** - holds the index of the next empty spot in the playlist
       - **song *arr** - an array of songs
     - Constructor
       - Sets **id** to 0 and creates a dynamically sized array of songs (**arr**) size **N** (where N is the inputted size)
     - Destructor

        - Frees **arr** with **delete[] arr**

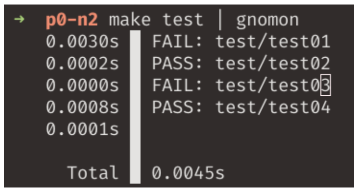
- Why are my data members private
    - There is no reason for other functions to change them

- For each function, examine the parameter list:
    - I add const *only* after my getter functions (**getName()** and **getArtist()**)
        - This is for const correctness; so my functions are const which guarantees that *any* members will not be changed in my getters
    - No parameters need const as we are passing in copies anyways (as I do not use pass by reference since my playlist class holds all the info I need)

4. **Test Cases**
    - Tested memory leaks using valgrind
    - Wrote a bash script to automatically run tests (and made it a makefile target)
    - Made test cases to test edge cases; e.g. inserting when the playlist is full

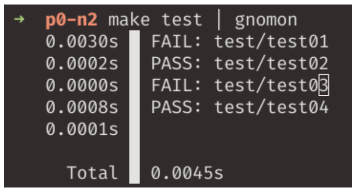
Note on segfaults:
I had a lot of difficulty at the beginning with “random” segfaults. After lots of debugging (with help from Mike and the TAs), I found that the issue was that I had called a default constructor followed by a parameterized constructor. While this itself is not an issue, my deconstructor was called when the default constructor is out of scope which happens when the object is rewritten with my parameterized constructor. Checking through gdb, it looks like the deconstructor was being called after the parameterized constructor causing the memory I had just allocated to be deleted. This means that inserting memory would cause segfaults but *reading* would be completely fine (only invalid read).

5. **Performance Decisions**
    - Pragma once in my header file causes it to be included only once in compilation
    - I wrote a hashmap implementation to prevent an O(n) insert
        - As when inserting you have to check if the song is already in the playlist
        - A hashmap can do this on average O(1)
    - Note: a hashmap is virtually no faster than looping as **std::cout** is the bottleneck see image below (no, they are not the same image despite the cursor position)
        - Thus, I commented out the hashmap implementation as it wastes memory for no real performance gain



```
➜  p0-n2 make test | gnomon
   0.0030s ▌ FAIL: test/test01
   0.0002s ▌ PASS: test/test02
   0.0000s ▌ FAIL: test/test03
   0.0008s ▌ PASS: test/test04
   0.0001s ▌

     Total ▌ 0.0045s
```



```
➜  p0-n2 make test | gnomon
   0.0030s ▌ FAIL: test/test01
   0.0002s ▌ PASS: test/test02
   0.0000s ▌ FAIL: test/test03
   0.0008s ▌ PASS: test/test04
   0.0001s ▌

     Total ▌ 0.0045s
```