|  |  |
| --- | --- |
| Program: | CPA3 |
| Course: | INFO-3135 “Data Structures & Algorithms” Fall 2025 |
| Professors: | Michael Feeney |
| Project # 1: | Managing your “Songify”\*\* music collection. |
| Weight: | 50% of your “project” mark, or 15% overall |
| Due Date: | Sunday, November 9th, 2025 @ 11:59 PM |

Description and Purpose

“Spotify” is a music streaming app (<https://en.wikipedia.org/wiki/Spotify>) whereas “Songify”\*\* is... OK, the idea is it’s basically the same thing, but we’re not going to call it “Spotify”. If you’re unfamiliar with “Spotify”, it’s basically like an iPod or Apple Music or YouTube Music or WinAmp or some ancient MP3 player that you found at the thrift store and picked up because it’s “so cool and retro!”.

You’re going to create a super amazing application that keeps track of people (users) and the songs they like, by loading information from a set of files, generating example data, and using containers and fundamental sorting and searching code that you have created yourself. STL (or boost or whatever) are not allowed.

While you *can* make an interface for this (like a GUI or text-based or whatever), you *aren’t* marked for that. Instead, you are given some class shell/starter code (specifically, a set of classes/structs with certain methods) that you are creating the implementation for.

The idea here is that you’ve been tasked to make just a *part* of a much larger application. Your code will be inserted into a larger, existing project and it is expected to work properly and without incident (catastrophic failure, crashing, data loss, etc.)

You are being marked on how correctly you implement the *functionality* of the API using your code.

**Note:** Because these class/struct interfaces define the API, you *can’t* change them. For instance, if you decide that the **postalCode** variable in **cPerson** should really be a **string** instead of a **char** array of 6 values, in your opinion, then... well you can’t do that because the rest of the application is expecting a **char** array of 6 values.

Likewise, if you add other methods they won’t be called by the hypothetical existing application.   
I mean, you *can* add them if you’d like (maybe they are being called by your own methods), but they will be completely ignored.

The [TL;DR](https://en.wikipedia.org/wiki/TL;DR) of this project:

* A “person” generator (cPersonGenerator) ***(you are given this)***:
  + This is a class that will read information from several external data sets of provided US census files, in order to randomly generate a number of people.
  + This will generate reasonable “person data” that you can use for testing.
* A “music” generator (cMusicGenerator) ***(you are also given this)***:
  + This class will read information from an external data file (from the “billboard top 100”) to generate a playlist/library.
  + This will generate reasonable “song and artist” data that you can use for testing.
* A set of “Songify” classes (you are given these too):
  + cSongify class: the main class that you create implementation code for
  + \*\*cSong class: the “song” class used in the cSongify API class methods
  + \*\*cPerson class: the “person” class used in the cSongify API class methods
    - \*\*NOTE: these don’t have to be the formats of the song and person data that you use internally to your code (in fact, you might want to make your own types), but they are the classes that have to be used in the API. In other words, you can’t make changes/additions to these classes.
* The code to go with the “Songify” structs/classes (what you make):
  + Will allow people to be added
  + Allow people to be removed or edited
  + Look up people in a number of ways
  + Add and removing songs (title, artist, etc.) on the user’s playlist/library
    - they only have one library/playlist (think of this as the songs they’ve listened to/rated)
  + Keeping track of ranking of these songs (for the *user*, not globally)
  + Perform a number of searches based on various criteria
  + Allow data to be updated (song details, user details, etc.) and have it reflect across the entire system. For example, if Taylor Swift changes her last name to “Kelce” after she gets married (like that’s going to happen...) then all the playlists should reflect this. i.e. if I change the *song* information (or say, delete a song) then this change should automatically be reflected when I next retrieve any user’s song library.

Your classes will be added to a Win32 (windows API) 64-bit application, running in both Debug and Release mode, on Windows 10, which will “exercise” (“call”) these various methods on these classes. i.e. be sure that your classes will compile and link when placed into a 64-bit Win32 console application.

While you are encouraged to write your own application that will use these classes, you won’t be marked on this; only the classes and structures that you implement (with the interfaces described below) will be marked.

**You will submit:**

* Your amazing **Visual Studio solution.**
* A **short video highlighting your code** (more details later in the document)

Details

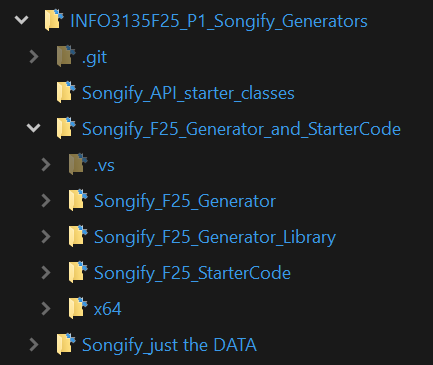
* You need to use the classes in the **Songify\_API\_starter\_classes.7z** archive (**cPerson**, **cSong**, and **cSongify** classes).
  + You may add anything to these classes that you’d like, but the **original interface and variable names MUST match**. i.e. the rest of the code assumes they have these public variables and/or methods exist as originally provided and the data can be accessed.
  + For **cPerson,** access is just through the public variables except for the **getSongifyUniqueUserID()** method that’s used to identify users. i.e. any getters/setters that are added will be ignored.
  + For **cSong**, I’ll be accessing the public data members except for the **getUniqueID()** method, not any getters/setters you might want to add.

|  |
| --- |
| * **The TL;DR is:**    + **You need to use the interfaces described in the cSongify.h, cSong.h, and cPerson.h files.**   + **You can *add* whatever stuff you want to that (but I won’t be calling them, of course).**   + **The ideas is that I should be able to replace any one of these, from any of you, interchangeably, in any testing code I’ve got.** |

* The methods/functionality of **cSongify** is where you’ll do most of your coding:
  + The testing code will create an instance of your class, then call the various methods.
  + The “starter code” has additional notes about what the various methods do.
* To generate test data, you are provided two classes cPersonGenerator and cMusicGenerator that you can use ***if you want to***. These parse data from various online sources (Billboard magazine, popular baby names, San Francisco street names, and surnames).
  + **Songify\_Music\_and\_Person\_Generator.7z** contains all the code and data (it’s big)
    - It’s also here: <https://github.com/FeeneyCodes/INFO3135Fall2025_Project_1_Songify_Music_and_Person_Generators>
    - All the data is in the “Data” folder within the Visual Studio project folder.
  + While there’s a lot of methods, etc., in the classes, you only need to use a couple of them:
  + To use cMusicGenerator:
    - Call: LoadMusicInformation()
    - Call getRandomSong() or getRandomMadeUpSong() to return a test song
  + To use cPersonGenerator:
    - Call: LoadCensusFiles()
    - Call generateRandomPerson() to return a test person.
  + The code that will exercise *your* code uses these two classes.

**Projects and files you’re given and what to do with them:**

A repository of the files are located here: <https://github.com/FeeneyCodes/INFO3135F25_P1_Songify_Generators>

There are a number of solutions & projects here, most of which of which you don’t explicitly *need* but it depends on how you want to go about getting test data.

Here’s a breakdown of what’s here:

**+ Songify\_API\_starter\_classes** [folder]

**+ Songify\_API\_starter\_classes.7z**

**|** (Just the class files - mostly headers)

**|** (These are the same: one is a folder, one an archive)

**|**

**+ Songify\_just the DATA** [folder]

**+ Songify\_just the DATA.7z**

**|** (the original data files the generator uses, in case you're interested)

**|** (These are the same: one is a folder, one an archive)

**|**

**+ Songify\_F25\_Generator\_and\_StarterCode** [folder]

**|**

**+--+ Songify\_F25\_Generator** [folder with project]

**|** (Source code for data generator + data)

**|**

**+ Songify\_F25\_Generator\_Library** [folder]

**|** (Same as above, packaged as a library)

The one you want

**|**

**+ Songify\_F25\_StarterCode** [folder]

(Example starter solution, with library and data)

Songify\_F25\_Generator.sln opens all three of these projects.

Songify\_F25\_StarterCode.sln only opens the Songify\_F25\_StarterCode.

**What do you do with all of this?**

You can copy/move the Songify\_F25\_StarterCode.sln and Songify\_F25\_StarterCode folder to another location and work with that.

The libraries it uses are generated from the source code in Songify\_F25\_Generator, so if these are changed, you'd need to rebuild Songify\_F25\_Generator\_Library to update the .lib files, then you'd need to copy the .lib files (located in the x64 folder) to the "lib" folder in Songify\_F25\_StarterCode.

Note that you \*don't\* have to include these .lib files or the generator source code, but the tool that will exercise your code will be using them.

i.e. you could use the generators to produce some data to use any way you'd like (from a text file, etc.).

**A note about the methods that returns “lists” of things (like GetUsersSongLibrary(), etc.):**

These return basic heap arrays that are ***copies***of the data contained inside the cSongify class.

Because you are making your own data-structures/containers, they aren’t returned by a standard container like std::vector or std::list.

You’ll need to allocate a heap array, then ***copy*** the values to this array and return it.

The “return” is really via reference, not literally “returned” by the method.

(the actual return value is for error conditions)

For example, if **GetUsersSongLibrary()** is called, the caller would do something like:

**cSong\* pSongArray = NULL;**

**unsigned int libraryArraySize = 0;**

**yourSongify.GetUsersSongLibrary(1234, pSongArray, libraryArraySize);**

*The caller now has an array with* ***copies*** *of the song data*

Both of these values, including the pointer, are passed *by reference* so the code in the method would have to do something like:

**libraryArraySize =** *..do-whatever-code-you-need-to-get-the-number-of-songs..*

**pSongArray = new cSong\*[libraryArraySize];**

*...COPY all the songs into this new array*

So you are ***NOT*** returning any direct pointers/access to the data internal to the cSongify class.

The ***caller***is then responsible for freeing up this array (that the method created).

i.e. you don’t have to worry about calling delete on the pSongArray array.

**Internal data structures/code restrictions:**

* You can **\*NOT\*** use the "**auto**" keyword (nor can you use a #define/typedef to circumvent this). I will use find and replace to change “auto” to “HelloKitty”, then try to recompile. If it doesn’t compile/build, then you get a mark of zero. **This is absolutely non-negotiable.**
* You can **\*NOT\*** use any variant of the STL containers or algorithm libraries, boost, or any other library (standard or otherwise), other than your own. In other words:
  + You **can** use any STL *streaming* (iostream, fstream, stringstream, etc.) or *string* libraries, as well as the C++ 11 random library (*though regular rand() is completely fine for this project*).
    - **If there is another *standard* library you’d like to use, please check with me first.**
    - **Keep in mind that the main intent of this course is for you to make your own data structures and algorithms, so if it’s a library that’s really a “data structure” and/or basic algorithm, my answer is going to be “no”.**
  + You can **\*NOT\*** use any of the STL vector, list, hash, map, (container) libraries.
  + You can **\*NOT\*** use any STL “algorithm” (or equivalent) libraries
  + Note that part of the interfaces can return all/some of the data as a regular “C” array (i.e. a pointer). This is ***not*** to suggest that you implement everything internally as an array, just that you will have to copy this information into an array to “return it” (by reference).
* You must handle your internal data in any combination of doubly-linked list, priority queue, or smart array (vector) data-structures/containers.
* You may use any kind of sort you want *except* bubble sort.
* I barely care about your sorting algorithm, so long as it works (i.e. sorts), so:
  + You can “go nuts” and make some “sexy” sort algorithm (radix, dilithium crystal phase-inverted sort, etc.), or you could, you know, just do something simple like use quick sort. Whatever.
  + You can also move items from one container to another, effectively causing a sort.
    - Think about it: adding an unsorted vector of items into a priority queue, prioritized by last name, will cause it to be sorted by last name. ...and maybe that’s even faster?
  + You can keep things sorted as you go by having multiple containers in use at the same time, but that are sorted in various ways.
  + You can keep “indexes” of items that are sorted (like databases do), so there’s a container sorted by first name that refers to “IDs” (or even pointer locations) in the “main” list of items. Then there’s another index sorted by last name, etc.

What you will submit (and also some marking notes):

* **Your entire Visual Studio solution** (**PLEASE** remove the “extra” files from it, making it smaller), and compress it.
* **A short recorded video** showing me where you’ve implemented (and used) your linked list and smart array. Use OBS/FRAPS/whatever to record this.
  + **I only need to see your screen and hear your amazing “radio” voice.**
  + Just make this very brief, something along the lines of “Here’s where I put my songs into my linked list when AddSong() is called” and “Here’s where I sort the values of my smart array when FindUsersFirstName() is called”.
  + If can be any format, as long as it plays in the VLC player (for Windows/Linux).
  + **This is non-negotiable.** While I heard some people say they don’t have a web cam (which you are required to have, anyway) there’s no way you don’t have a microphone and/or can’t record your screen. And you can also record your screen from your phone, if needed.
  + **TO BE CLEAR:** I **\*don’t\*** want an image from your webcam – I want a recording of ONLY your screen, showing visual studio, with your code in it.
* **If you are missing either of these, your submission won’t be marked and you will receive a mark of zero.**
* Any additional notes you think I need to mark this.
* You’ll mainly be marked by:
  + Your code will be placed into another solution that will “exercise” (call) your code, and compare it with the expected results.
    - This will do basic things like adding/removing/updating users, songs, etc.
    - It will also check for song lists and ratings.
    - It’ll check for updates, like if a song title is updated, that should be reflected on every user’s playlist.
    - It’ll check for errors, like if one user rates a song, that doesn’t impact any other user’s rating of the same song.
  + You code will be built - if it won’t build, you might be “dead in the water”.
    - No build = no marks.
  + Additional inspection of your code looking for:
    - Violations in what you’re not supposed to do (auto, any STL containers/algorithms, etc.)
    - Looking at your data-structures (and that you’re actually using them at some point)

Marking scheme:

* 40%: The rudimentary implementation of the methods.
  + Unless these completely screw up, you’re likely to get this mark.
  + For example, if **AddSong()** is called then **GetSongToPlay()** should return that song.
  + This *doesn’t* take into account the interplay between the rest of the class though, like if you add a song to someone’s playlist then delete that song, what happens then?   
    (that’s the next 60% of the marks...)
* 60%: did you implement these correctly?
  + You can get a sense of the mark “breakdown” or “weighting” by how many methods there are and now complex they are: **GetCurrentSongNumberOfPlays()** is going to be “worth” more marks than **AddSong()** for instance (because it’s more involved)
  + Are the more complex methods working, like is the **GetUsers()** *actually* sorted?
    - If a user name is updated, does the **GetUsers()** sort reflect this?
  + Are the interplays between the structures working?
    - If a user’s song rating is changed, does it only impact *that* user alone? Or does it do something strange/unexpected/wrong?
    - If a user is deleted, does it do anything to anyone else’s song library?
  + How does it handle expected “errors” (like looking for a person that’s not there, etc.)
  + How appropriate is your choice of implementation?
    - Does your search/sort take “forever”? (It should take a moment on a typical modern computer.) Are you trying to do a quick sort on a linked list? Are you sorting the entire container with every “Add”? Every change in rating? Does deleting a song take 30 seconds? (it should happen instantly). Does adding duplicate songs (there *are* duplicates in the billboard file, of course), mean adding hundreds of instances? (it shouldn’t). Does the unique ID change every time you “touch” the container? (it shouldn’t, since it’s “unique”, right?) Does changing a song title crash the application? If I run it for a few minutes, does it use up all my RAM? Does it eventually crash?
    - Think about if this was a ***real*** song library app - What would your performance/functionality expectations be? Then go with that. Are you OK with waiting 15 seconds if you click the “+” (“like”/”add”) in Spotify? Does Spotify repeatedly and randomly crash for no reason? Didn’t think so...
    - This indicates that you know when to implement what, not just that you did the bare minimum. Like you *could* store everything in linked-lists, and do a bubble sort for everything, but that’s not really the goal, right?

Project Corrections

If any corrections or changes are necessary they will be posted to the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project. Additional resources relating to the project may also be posted.

75/10-year old “squinty eye” plagiarism test:

I have very little tolerance for plagiarism, but many students are unclear about what it is.

Basically, it’s submitting somebody else’s work as your own.

There is sometimes some confusion over this because you could argue nothing is actually “unique” (see: <http://everythingisaremix.info/> for a fascinating overview of this).

The whole point of assignments/tests/projects in this course (or any course, really) is to try to see if you are actually able to ***do*** the coding that’s asked of you. In other words: How competent are you? Handing me someone else’s code and/or making a trivial change isn’t good enough.

Also, it’s illegal:

* <http://www.plagiarism.org/ask-the-experts/faq/>
* <http://definitions.uslegal.com/p/plagiarism/>
* <http://en.wikipedia.org/wiki/Plagiarism>
* <https://www.legalzoom.com/articles/plagiarism-what-is-it-exactly>

In other words, I’m not going to be drawn into a giant debate over how “different” your code is from mine or anyone else’s, if any sensible person (including me) would conclude that the code/application is pretty much the same thing, then it is. It is up to my discretion to decide this.

* While you may freely “borrow” my (or anyone other) code ***but*** your code should be “sufficiently” different from mine (you might want to replace the word “sufficiently” with “significantly”).
* In other words, you *cannot* simply use an existing game engine (or part of a game engine) to complete this assignment; it should be either completely new of **significantly** modified.
* How will I determine this?
  + If I showed your application and/or your source code to either my pragmatic 83-year-old mother, or a typical 10-year-old, or even some random person walking down the hallway (i.e. a non-expert), and they looked at it, tilted their heads, squinted their eyes, and said “you know, they look the same,” then they ***are*** the same.
  + Another test would: How much time it would take for a "competent programmer" (for example, *me*) to make the changes you are submitting? The point here is that I don’t “care” if you tell me “But it took me *weeks* to make the changes!” Fine, but if I can make those same changes in 10 minutes, then not a lot of work has been done (certainly **not** sufficient work – these projects should show take **days** of work having been done).