|  |  |
| --- | --- |
| Program: | CPA2 |
| Course: | INFO-6023 “Game Algorithms & Gems” Winter 2024 |
| Professors: | Michael Feeney |
| Project # 1: | Managing your Snotify\*\* music collection. |
| Weight: | An equal portion of your 60% “project” mark  (i.e. all projects are equally weighted) |
| Due Date: | Saturday, February 24th at 11:59 PM  (Last day of “break/reading week”) |

Description and Purpose

\*Snotify is very similar to Spotify. OK, it’s pretty much the same thing, but with a funnier name.

You’re going to create a super amazing application that keeps track of people 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.

While you can make an interface for this (like a menu or whatever), you aren’t marked for that. Instead, you are going to be marked by how the API works, and how you implement the functionality internally.

The TLDR is your project will be in three (3) parts:

* A “person generator”:
  + 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.
  + These people are then added to the 3rd part (“Snotify”).
* A “song list” generator:
  + This class will read information from an external data file (from the “billboard top 100”), in order to generate a playlist/library for.
  + These playlists (with songs, artists, etc.) will be added to the 3rd part (“Snotify”).
* “Snotify” (like Spotify):
  + 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)
  + Keeping track of ranking of these songs (for the user)
  + Perform a number of searches based on various criteria  
    (and output this data to a file in a specific format)
  + Perform a number of sorts based on various criteria  
    (and output this data to a file in a specific format)

Your classes will be added to a Win32 (windows API) 64-bit application, running in both Debug and Release mode, on Windows 10 or 11, 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 **cPerson** and **cSong** classes that are in the 7z file with this project (the cPerson class is basically the one we’ve used in the lectures/help sessions).
  + You may add anything to this classes that you’d like, but the **original interface and variable names MUST match**. i.e. the code I’m using assumes they have these public variables and/or methods and that I can access that data.
  + For **cPerson** I’ll be accessing the public data and the **getSnotifyUniqueUserID()** method, *not* any getters/setters you might want to add.
  + For **cSong**, I’ll be accessing the public data except for the **getUniqueID()** method, not any getters/setters you might want to add.
* You need to implement the details of the following classes:
  + cMusicGenerator :
    - Used to load the initial song data to be added to the cSnotify class instance.
    - You can either implement this class or inherit from this class. If you inherit from this class, remember that I will be using this class as the base class (like I will instantiate your class, but I will only have access to the cMusicGenerator public variables and methods, so keep this in mind).
  + cPersonGenerator :
    - Used to load the initial person data go add generated users to the cSnotify class instance.
    - You can either implement this class or inherit from this class. If you inherit from this class, remember that I will be using this class as the base class (like I will instantiate your class, but I will only have access to the cPersonGenerator public variables and methods, so keep this in mind).
  + cSnotify :
    - The main class.
    - You can either implement this class or inherit from this class. If you inherit from this class, remember that I will be using this class as the base class (like I will instantiate your class, but I will only have access to the cSnotify public variables and methods, so keep this in mind).
* The “starter code” has additional notes about what the various methods do.

**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 me “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-lined list, smart array (vector), map (tree based) or hash map.
* You can’t just use my code from class. I mean there’s only “so many ways” to code things like a tree or a smart array or whatever, but it has to be reasonably clear that you’ve made “your own” structures.
* I could 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, use quick sort or even bubble sort. Whatever.
  + You can also move items from one container to another, causing a sort.
    - Think about it: adding an unsorted vector of items into a map, indexed by last name, will cause it to be sorted by last name. And it’s very likely that it’s faster.
    - Keep things sorted as you go. Like have many copies of the same data, sorted.
    - Keep “indexes” of items that are sorted (like databases to), 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.
* You **CAN** use things more “fundamental” parts of the STL like std::string, file/console I/O, etc. just not the containers and algorithm parts.
  + The idea is that you “aren’t using the STL at all”, so don’t have access to it.
  + However, while having you implement your own string classes would be fun, that seems like “too much” considering everything else you have to do.
  + Same with the std streaming I/O – that would force you to use the C functions, which don’t really help you learn anything about making your own containers, etc.

**Loading the census data:**

There are three files, one with “first” and “middle” names, one file for surnames (last names) and one for street information. You will have to load all three of these:

**First and Middle names:**

* The “first” and “middle” names are taken from the "Popular Baby Names" found here: <https://www.ssa.gov/oact/babynames/limits.html>
* The names.zip file contains information for 1880 to 2019, where “**yob1993.txt**” would be the list of names from the year 1993.
* Each line is in this format: Name, Identified Gender, Number of babies with that name, so:
  + “Emma,F,13312” would mean:
    - There were 13,312 babies born with the name “Emma” identified as “Female”
  + Note that there are some entries with the same name, but are also “M” (Male):
    - There’s a “Taylor,***F***,13692” and also a “Taylor,***M***,2309”
    - The person class has three identified genders as well as “prefer not to say”
    - When you randomly pick a person, you need to keep a few things in mind:
      * Some names can be male or female *if* they appear as both in the file
      * If they don’t (“Sheena” is an example, which is only female), then you can’t identify them as male.
      * However, *any* person can be “non-binary” (there is no “non-binary” in the file)

**Surnames/Last names:**

* The last name/surnames come from "Frequently Occurring Surnames from the 2010 Census" from here: <https://www.census.gov/topics/population/genealogy/data/2010_surnames.html>
* This contains two files, but you will be using the “**Names\_2010Census.csv**” files.
* This file has a lot of data, most of which you can ignore, but the 1st few items are useful to us:
  1. name : the name, eg. “SMITH”, “DAVIS”, etc.
  2. Rank: the popularity, so 1 would be the most popular
  3. Count: Number of people with that name
     + “Smith” has 2,442,977 meaning that there are over 2 million people named “Smith”
  4. prop100k: Population per 100,000 people.
     + “Smith” has a value of 828.19, meaning 828.9/100,000 people are named “Smith”
     + This gives you a “chance” of being named “Smith”, too:
     + 828.9/100,000 = 0.8289 % of the population is named “Smith”
  5. cum\_prop100k “cumulative” proportion (which isn’t super useful, but it’s essentially the total ratio as we go, so the 1st 10 names make up 4,902.9/100,000 or 4.902% of the population)
  6. pctwhite, pctblack, pctapi, pctaian, pct2prace, pcthispanic all refer to the identified race, which we will ignore.

**Street Names:**

* The "Street Names (San Francisco)" from here: <https://catalog.data.gov/dataset/street-names-37fec/resource/d655cc82-c98f-450a-b9bb-63521c870503>
* There are two files, but you will be using the "**Street\_Names.csv**" file, in this format:
  + FullStreetName, StreetName, StreetType, PostDirection
  + This: "MISSION BAY BLVD NORTH, MISSION BAY, BLVD, NORTH" would be:
    - Street Name = "MISSION BAY"
    - Street Type = "BLVD"
    - Post Direction = “NORTH”

**Loading the song data:**

Billboard magazine used to have an API that would allow reading “top 100” information. Unfortunately, that API isn’t there, anymore.

However, a copy of this information was stored at “Billboard Hot weekly charts - dataset by kcmillersean - data.world” here: <https://data.world/kcmillersean/billboard-hot-100-1958-2017>

The zip file: **kcmillersean-billboard-hot-100-1958-2017.zip** contains a “data” folder with a “**hot\_stuff\_2.csv**” file.

Loading this file is just like the US census information, in that it’s a comma delimited file.

While there’s some interesting information in there, you only need “song” and “performer” values (the 4th and 5th items in each row), which has the song title and artist(s) name(s):

* **url** : Original link to the billboard.com site (which no longer works)
* **weekid** : The date the song was on the list in “year-month-day” format
* **week\_position** : Position on the chart that week
* **song** : Song name
* **performer** : Artist(s)
* **songid** : no clue what this is for (I’m guessing something billboard used?)
* **instance** : again, no clue what this is for (so please ignore it)
* **previous\_week\_position** & **peak\_position** & **weeks\_on\_chart** : Just what you think they are ☺

Note that songs often are listed in the “top 100” for more than one week, so there are a number of identical records (of song & artist) in the list. You will need to handle that by either loading duplicates or checking to see if the song has already been loaded (i.e. ignoring any duplicate entries).

**Loading the files:**

The files are “delimited” by a new line character at the end of each line, but the data is separated by commas.

This is a pretty standard file format.

There’s many ways to load this, but below is a simple way, using the **std::getline()** function.

The getline() will take a stream object (cin/fstream/etc) and a string. It will then read the entire “line” based on a single “delimiter”. If you don’t specify one, it assumes it’s newline (‘\n’).

Unfortunately, you can only specify one delimiter in getline(), so this will read the file in two parts, the 1st while reading the entire line, then the 2nd while reading the values with the ‘,’ character as the delimiter (i.e. for just *that* line it loaded).

#include <iostream>

#include <fstream>

#include <sstream> // String Stream

#include <string>

bool Names\_2010CensusCVSFile(void)

{

// Open the file

std::ifstream namesFile("Names\_2010Census.csv");

if (!namesFile.is\_open())

{

std::cout << "Didn't open file" << std::endl;

return false;

}

// name,rank,count,prop100k,cum\_prop100k,pctwhite,pctblack,pctapi,pctaian,pct2prace,pcthispanic

// SMITH,1,2442977,828.19,828.19,70.9,23.11,0.5,0.89,2.19,2.4

//

// - rank is how popular the last name is, like 1st, 2nd, etc.

// - count is how many people have that last name, so 2,442,977 people have last name "Smith"

// - prop100k is the ratio per 100,000 people. Smith is 828.19,

// meaning that there's a 828.19 out of 100,000 chance (0.82819% chance)

// that someone is named "Smith"

std::string theLine;

unsigned int lineCount = 0;

while (std::getline(namesFile, theLine))

{

lineCount++;

std::stringstream ssLine(theLine);

std::string token;

unsigned int tokenCount = 0;

while (std::getline(ssLine, token, ','))

{

if (tokenCount == 0)

{

std::cout << token << std::endl;

}

// Ignore the other parts of the line (if we only want the last names)

tokenCount++;

}

}

std::cout << "Lines read = " << lineCount << std::endl;

return true;

}

You will submit and marking:

* **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 thing I need to mark this.
* I’ll mark it by:
  + I’ll place your classes into another solution that will “exercise” (call) your code, and comparing it with the expected results.
  + I’ll attempt to build your code, and if it won’t build, you might be “dead in the water”.
    - No build = no marks.
  + I’ll scan through your code looking for:
    - Violations in what I’ve asked you (auto, any STL containers/algorithms, etc.)
    - Seeing if you have a linked list and a smart array (and that you’re using them at some point)
  + Using a different/modified set of data. It will be the same format, etc., just that I’ll know the data that’s in there.
  + I’ll run my code, which generates songs and users, adds/edits them, and sees the results.

**Marking scheme:**

* 10% : cMusicGenerator
* 10% : cPersonGenerator
* 80% : Snotify class:
  + 40% The rudimentary implementation of the methods.
    - You can get a sense of the “breakdown” by how many methods there are.
    - Unless these completely screw up, you’re likely to get this mark.
    - For example, if I call AddSong(), then immediately call GetSong() I should expect that to work.
    - This *doesn’t* take into account the interplay between the rest of the class, thought, like if you add a song to someone’s playlist, then delete that song: what happens then? (that’s the next 60%) 🡪
  + 60% did you implement these correctly?
    - Are the more complex methods working, like is the GetUsers() actually sorted?
      * If update a user name, does the GetUsers() sort reflect this?
    - Are the interplays of the structures working?
      * If I change the rating of a song for a user, does it only impact *that* user alone? Or does it do something strange/unexpected/wrong?
      * If I delete a user, 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 be instant). 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 song library thing: What would your performance/functionality expectations be? Then go with that. You OK with waiting 15 seconds if you click the ♥ in Spotify? Spotify repeatedly and randomly crashing 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 showing me your either lazy and/or don’t really understand what’s going on.

Additional requirements:

* While you may freely “borrow” mine (or anyone other) code ***but*** your code should be “sufficiently” different from mine in terms of the output on screen. See the "plagiarism" test, later in this document, for more details.
* I am more concerned with it working and meeting the requirements, rather than “style” or anything like that. In fact, other than the basic requirements (no “auto”, etc.), and the general functionality of your data structures, I won’t really be paying much attention to that. So you should focus on it meeting the interface requirements (class & method names, etc.), then focus on it working as expected.   
    
  Suggestions:
  + Initially Make smaller data sets and load those. Then see if it handles larger sets.
  + Run your code often, gradually building functionality. Avoid trying to “design it all” at the start. This is because you don’t know what you really need to focus on, and are understanding the requirements as you go.
  + Develop in Debug mode, then when it’s working (or a part is working) switch to Release to see if it still works (Code often “breaks” when switching from Debug to Release).
* you can **\*not\*** use the "**auto**" keyword (nor can you use a #define/typedef to circumvent this)
* 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
  + You can **not** use any of the STL vector, list, hash, map, (container) libraries.
  + You can **not** use any STL “algorithm” (or equivalent) libraries
* Related to plagiarism/cheating and not doing much work:   
  + If you simply submitted the in class code, then you invested zero time, so you did no work, so you get a mark of zero (in that case, it is a clear case plagiarism/cheating, and an academic offence would also be submitted).
  + If you took the in class code and made some trivial changes – like simply changed the names of the method calls - you might not have actually "cheated", but you did essentially no work.   
      
    The tests here would be:   
    - "How long would it take me - your instructor - to make those changes?" If it's something that would take me 5 minutes, don’t expect any marks for that.
    - If I picked a random person (non programmer or programmer) and had them compare your code with the in-class code (or anyone else’s code), and they determined they were “the same” (or close enough), then they *are* “the same”, and you have cheated.
  + If you code does not build (i.e. linker error) and run (i.e. no crazy run-time crash that is unexpected), I may investigate this further, but only if there is some simple problem and/or *very* slight and/or *very* obvious (and easy to fix) configuration error or last minute typo. This includes accidentally including other libraries or dependencies that I don’t have (like boost).  
       
    Since it can't run, you would get a mark of zero. There are no ‘part marks’ in this case.

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” mine (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).