CS 3240, Fall 2017
Assignment 0 README
Due: September 14
Part 2 of this assignment will be due Sept 21

You have been given a file "SongCSV.csv" which contains comma separated data on separate lines. You are to read that file and create a struct for each song that includes: Artist, Name of Song, Album Name, Duration, Year of Release, and Hotttnesss.

For the first song in the file

1,"SOCJJDX12A8C13E745",222209,"Aswad vs. The Rhythm Riders","ARP39OU1187FB4D543",,"West London England",,"Aswad",0.0,258.42893,1,0.69,199.826,nan,7,0.32,"Rudeboy",0

the relevant fields are Aswad, Rudeboy, Aswad vs. The Rhythm Riders, 258.42893, 0, and nan(enter 0.0). Artist, Name of Song, and Album Name is stored as char *,Duration is a float, Year of Release is an integer, and Hotttnesss is a double

You will dynamically allocate memory for each struct as the file is processed. An array of pointers to these will be managed so that searching can be accomplished (binary search because you sorted the pointers).

You will fopen() the file, and then use the I/O operations from Chapter 5 (standard library) in Stevens to process the file (fgets(), fclose(), feof() etc). You may use strtok_r() from the string libraries, as well as atoi() and atof().

Think of this program as a step towards something larger. Demonstrate that your program works by searching the array for a given song and outputting the struct data. Make a typescript (use utility 'script') of this performance by searching with the following keys:

For the output you may use fprintf() to stdout. We will test this ourselves, but for the moment take user input (the song name) and use it to search the sorted structs using … ! … binary search.

This assignment will ultimately be on the jazz server … a work in progress. I hope we can have all of this taken care of quite soon.

Basic Algorithm Flow:

1Open the csv file

2 Extract lines, one at a time.

> Put relevant fields (see above) into a struct which is then put into an array of structs. Please note that you are required to allocate memory for structs dynamically.

3 Sort the array of structs by song name. (You may have been doing this in step 2)

4) Create user interface to search for songs by song name Display relevant data to user in a neat format Exit only when user enters ZZZ (not 'Quit' or 'q' or 'Q' or 'exit' or 'Exit' or 'EXIT' or 'Leave'' or anything other than ZZZ)

We have provided a make file for you and a sample testing input file (we will cleverly use a different one for actual testing). If your program does not work with 'make && make test' we will not be able to test your program, and you will not like your grade.

Also, would you please push *only* the source code and typescript of performance. Please *do not* push back the various files that have been provided to you (e.g. syllabus).

Discussion

The intent of this assignment and the next is to immerse you in the C / Unix system programming environment with the good, bad, and ugly of it all. A number of issues are worth mention (in no particular order here).

0) **YOU MUST HAVE A LAPTOP RUNNING UNIX**. Although you can survive with a VM , it offends me that you don't have something more permanent – just old fashioned. Have the wizards help you dual boot. Bring your machine to class.

I) **C and standard library issues**

1) Pointers – love the asterisk. Know how to declare, instantiate i.e. make the pointer point somewhere, dereference and assign values to the 'pointee'. Understand that because C as a language is limited the '*' shows up in parameter passing, strings (char *), and function returns of compound data structures (arrays, structs). Understand that arrays are synonymous with pointers and that pointers can be subscripted cuz why not ?
2) dynamically allocated memory via malloc(), calloc(), realloc() requires using pointers.
3) Structs and pointers – fields are referenced with . or -> depending on how you have access to the struct. Structs can be copied wholesale.
4) The string functions all use char * references *with a null terminator.* strlen() does not count the null but allocating memory for a strncpy() requires that extra byte. Be careful with strtok_r(). Use the 'n' versions of string functions (strncpy() vs strcpy()), Use the _r versions ('reentrant) of functions.
5) Standard library I/O use the 'f' library routines built upon the non-f system calls: fopen() vs open(), buffered streams vs unbuffered references via file descriptors; formatted fprintf() vs unformatted write().

II) From Source to Executable the Unix Way

The steps of creating programs: cpp, gcc (translate), gcc (load via ld), and then execute. *Make* files automate the above. Source code is 'pre-processed' to take advantage of #include directives and good software practice

III) Use The Utilities

Use utilities such as script, make, valgrind , and gdb – NOW. Your life will be easier if you invest in minimal mastery of these tools. *Make* automates command line source code management. Valgrind insures you don't have pointers going astray and often reveals bugs. Gdb allows you a debugger. Script lets you document program behavior. Each has its idiosyncracies.

IV) The Assumed Stevens Working Environment

You may use an IDE if you wish but we will assume command line capability. Use gedit for screen editing capabilities. Be able to navigate and reproduce Stevens code.

The Stevens environment. "apue.h" and the companion source code "error.c" are used throughout Stevens' source code. Use these because then you can reproduce his examples. I will provide copies of apue.h, error.c, and all of the source code in the text. You *must* test all system call return values so use Stevens' error routines as a habit. See Ch1 for the errno, perror() scheme provided in all Unix environments.