

Assignment 1

(Part 2 after Assignment 0)

Due TBD

PLEASE HONOR NAME CHOICES

You are to use the functionality of Assignment 0 to make a permanent(ish) database of the ordered country data you have extracted from Songs.csv. You are to do this with two programs:

The first is BuildDataBase -- this will put the structs you have extracted from the original file into a file called BinarySongData. In addition there will be a 'directory' that contains the correspondence between the song names and the offset of that song struct. This directory will be sorted. You will write the directory as a separate file. After BuildDataBase the above information has been created and the program terminates.

The second is UseDataBase - this will open the file(s) created above and allow a user to ask for a song name and be presented with the song information (formatted reasonably). For both of the above you MUST use Chapter 3 raw I/O system calls. NO 'F' WORDS (fopen, fwrite, fread, fseek, etc). However, you are allowed to use fprintf() for output.

You will create your own buffer for reading to and from files. The buffer size will be defined with a #define directive BUFFSIZE with a value that is 4096. The UseDataBase interface will prompt the user for a song name and exit when 'ZZZ' is entered. Same as before. You will submit a) the two programs above along with any headers you have created, which *MUST* be named BuildDataBase.c and UseDataBase.c.

Another makefile and sample input data file has been provided. Name your programs so that the command 'make && make build && make use' compiles and runs both programs.

Discussion

This assignment will clarify the distinction between a) system calls and libraries built on top of them ('standard library'), and b) buffered and unbuffered I/O. The previous assignment allowed, for example, the use of fgets() to extract lines from a file fopen(ed) as a 'stream' or FILE* object. This assignment will open() and reference the file using the integer 'file descriptor'. The use of stdout vs STDOUT_FILENO illustrates this distinction. You now must find the 'lines' in the file yourself. Think in terms of writing your own 'GetLine' which you must do by finding the '\n' delimited characters. Note that a line can span multiple buffer fills. Isolating that work will allow you to reuse much of the previous assignment.

The idea of a 'binary' file vs an 'ascii' file is also highlighted here. The data written to the database, and to the directory does not need to be read by humans so don't waste the bytes.

Please honor our name choices so we can test more easily.

