Assignment 3 (A3)

Due date: Friday, July 14, 2023 (11:59pm)
Submission via Git only

Contents

Programming environment

For this assignment you must ensure your code executes correctly on the *reference platform* (i.e., computers on ELW B238, which can be accessed using ssh) you configured as part of Lab 01. This same environment will also be used by the teaching team when evaluating your submitted work. You will have to make sure that your program executes perfectly on the reference platform. If your programs do not run on reference platform, your submission will receive 0 marks.

All test files and sample code for this assignment are available in the 'a3' folder of your git repository and you must use the git pull command to download a copy of the files:

```
git pull
```

Individual work

This assignment is to be completed by each individual student (i.e., no group work). You are encouraged to discuss aspects of the problem with your fellow students. However, sharing of code fragments is strictly forbidden. Note SENG 265 uses highly effective plagiarism detection tools to discover copied code in your submitted work. Both, using code from others and providing code to others, are considered cheating. If cheating is detected, we'll abide by the strict UVic policies on academic integrity: https://www.uvic.ca/library/help/citation/plagiarism/

Learning objectives

- Revisit the C programming language, this time using memory allocation and dynamic data structures.
- The starting point for A3 is a 'skeleton' C program. You need to study and modify this program appropriately to complete the assignment.

- Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits. Update your git repository after every major editing session to make sure that you don't lose your work.
- Test your code against the provided test cases.

music_manager.c: Descriptive analytics about songs data

a) In A3, music_manager is the same program we created for A2, but implemented in C with dynamic memory allocation.

b) Based on provided arguments and data files (i.e., datasets), music_manager will help us answer the following questions:
   - **Q1**: What are the top 10 songs released in 1999 with most 'popularity'?
   - **Q2**: What are the top 5 songs released in 1999 with most 'energy'?
   - **Q3**: What are the top 3 songs released in 1999 with most 'danceability'?
   - **Q4**: What are the top 3 songs released in 2009 with most 'popularity'?
   - **Q5**: What are the top 5 songs released in 2019 with most 'danceability'?

c) After each execution, your program **must produce a file named output.csv** that represents the answer to the question asked to program (i.e., arguments passed). The data in this file should follow a descending order based on a required column (i.e., 'popularity', 'energy', or danceability'). If two songs share the same value for the sorting column used, the items should follow a lexicographical order based on the 'song' column. This means that the (descending) order of elements (rows) in output.csv is determined by:
   - 'popularity', 'energy', or 'danceability'
   - Lexicographical order in 'song'

d) The most reliable way (and the only one encouraged) to test your program is to use the provided **tester** file which will validate the output produced by your program, given a particular question.

e) The program itself consists of several C source files and a makefile for build management:
   - emalloc[.c or .h]: Code for safe calls to malloc, as is described in labs\lectures, is available here.
   - Linky[.c or .h]: Type definitions, prototypes, and codes for the singly-linked list implementation described in labs/lectures.
   - makefile: This automates many of the steps required to build the music_manager executable, regardless of what files (.c or .h) are modified. This file can be invoked using the Bash command make.
   - music_manager.c: The main file of the program.

Restrictions

- To store song info, you need to allocate storage on the heap dynamically. Dynamic data structures must be used in the form linked-list manipulation routines (i.e., **arrays of songs are not permitted for this assignment**). The elements of the linked list can be struct types.
- You must not use program-scope or file-scope variables.

Testing your solution

- As with A2, this time the **tester** file will execute your program automatically, **but you'll have to compile it first with make**. You'll only need to pass the number of the question as an argument (e.g., `./tester 1`). If no arguments are passed to the tester file (i.e., `./tester`), it will run the tests for all the questions. Using a specialized library that compares the differences between .csv files, the **tester** file will describe the differences between the expected output and the one provided by your program.
- Refer to the example commands in the file "TESTS.md" for appropriate command line input. Your solution must accommodate all specified command line inputs.
- Make sure to use the test files provided as part of this assignment.
- Use the test files and listed test cases to guide your implementation effort. Develop your program incrementally. Save stages of your incremental development in your Git repository.
- For this assignment you can assume all test inputs are well-formed (i.e., exception handling is not required).
- **DO NOT rely on visual inspection.** You can use the provided **tester** file so you can verify the validity of your outputs in a simpler manner.

What to submit

The six files listed earlier in this assignment description (i.e., music_manager.c, emalloc.c, emalloc.h, list.c, list.h, makefile) plus any other files you use in your solution, submitted to the a3 folder of your Git repository.

Grading

Assignment 3 grading scheme is the same used for Assignment 1 (A1). Please refer to A1's write up.

Input specification

Please refer to A2's write up.

Output specification

Please refer to A2's write up.