Assignment 3

Due date: Thursday, July 14, 2022
Submission via Git only

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines (i.e., Senjhalla or its analogous environment for students with M1 MacBooks) you installed/configured as part of Assignment 0 as this is our "reference platform". This same environment will be used by the teaching team when grading the work submitted by the SENG 265 students.

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.

Any programming done outside of Senjhalla might result in lost marks or even 0 marks for the assignment. Make sure that **all** your submitted files are placed correctly in your **a3 folder**.

Hint: To verify whether you uploaded the files properly, please clone the git repository to a new directory on your computer and check that the files you intend to submit are there.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Sharing of code fragments is strictly forbidden. **Code-similarity analysis tools are used to check submitted work for plagiarism.**

Learning objectives

I. Revisit the C programming language, this time using memory allocation and dynamic data structures.
II. The starting point for A3 is a 'skeleton' C program. You need to study and modify this program appropriately to complete the assignment.
III. 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 loose your work.
IV. Test your code against the provided test cases.

This assignment: song_analyzer.c, using C's heap memory

You are to write an implementation of song_analyzer in C such that:

a) Based on provided arguments and data files (i.e., datasets), song_analyzer 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'?
   - **Q6**: What are the top 5 songs released in 1999 or 2009 with most 'energy'?
   - **Q7**: What are the top 10 songs released in 1999, 2009, or 2019 with most 'popularity'?

b) Similarly to A2, 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' (descending)

c) 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**).

d) 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 song_analyzer executable, regardless of what files (.c or .h) are modified. This file can be invoked using the Bash command make.
- song_analyzer.c: The main file of the program.

Relevant observations

- You must not use program-scope or file-scope variables.
- You must not use arrays. Use linked lists instead. The elements of the linked list can be struct types.

Testing your solution

- 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.
- For this assignment you can assume all test inputs are well-formed.
- Do not rely on visual inspection. Human eyes are poor at comparing white space. You can use the provided tester file (i.e., "tester.py") 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., song_analyzer.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 as follows. In general, straying from the assignment requirements will result in zero marks due to automated grading.

A grade: A submission completing the requirements of the assignment and is well-structured and clearly written. Global variables are not used. song_analyzer runs without any problems; that is, all tests pass and therefore no extraneous output is produced. An A-grade submission is one that passes all the tests and does not have any quality issues. Outstanding solutions get an A+ (90-100 marks), solutions that are not considered outstanding by the evaluator will get an A (85-89 marks) and a solution with minor issues will be given an A- (80-84 marks).

B grade: A submission completing the requirements of the assignment. song_analyzer runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written. Although all the tests pass, the solution includes significant quality issues. Depending on the number of qualitative issues, the marker may give a B+ (77-79 marks), B (73-76 marks) or a B- (70-72 marks) grade.

A submission with any one of the following cannot get a grade higher than B:

- Submission runs with warnings
- Submission has 1 or 2 large functions
- Program or file-scope variables are used

A submission with more than one of the following cannot be given a grade of higher than B-:

- Submission runs with warnings
- Submission has 1 or 2 large functions.
- Program or file-scope variables
- No documentation is provided

C grade: A submission completing most of the requirements of the assignment. song_analyzer runs with some problems. This is a submission that presents a proper effort but fails some tests. Depending on the number of tests passed, which tests pass and a qualitative assessment, a grade of C (60-64 marks) or C+ (65-69 marks) is given.

D grade: A serious attempt at completing requirements for the assignment (50-59 marks). song_analyzer runs with quite a few problems. This is a submission that passes only a few of the trivial tests.

F grade: Either no submission given, or submission represents little work or none of the tests pass (0-49 marks). No submission, 0 marks. Submissions that do not run, 0 marks. Submissions that fail all tests and show a poor to no effort (as assessed by the marker) are given 0 marks. Submissions that fail all tests, but represent a sincere effort (as assessed by the marker) may be given a few marks

Additional Criteria for Qualitative Assessment

Since SENG265 is a Software Engineering course, **ONLY** solutions that comply with the criteria mentioned below and with other relevant aspects at discretion of the evaluator will be considered as A+ submissions (i.e., will get full marks).

- **Documentation and commenting:** the purpose of documentation and commenting is to write information so that anyone other than yourself (with knowledge of coding) can review your program and quickly understand how it works. In terms of marking, documentation is not a large mark, but it will be part of the quality assessment.
- **Proper naming conventions:** You must use proper names for functions and variables. Using random or single character variables is considered improper coding and significantly reduces code readability. Single character variables as loop variables is fine.
- **Debugging/Comment artifacts:** You must submit a clean file with no residual commented lines of code or unintended text.
- **Quality of solution:** marker will access the submission for logical and functional quality of the solution. Some examples that would result in a reduction of marks: solutions that read the input files several times, solutions that represent the data in inappropriate data structures, solutions which scale unreasonably with the size of the input.

Input specification

- All input test files are in CSV format and are based on the "Top Hits Spotify from 2000-2019" dataset in Kaggle.com.
- The metadata (i.e., description of each column) for the input datasets can be found online in the 'Content' section of the dataset description.

Output specification

- After execution, your program must produce\create the following file:
  - **A. output.csv**: This file will contain four-dimensional data (i.e., a table with four columns) that represents the answer to the question passed as an argument to the program (e.g., **Q1**, **Q2**, **Q3**, **Q4**, **Q5**, **Q6**, **Q7**).
    - The first column refers to the name of the song's artist and must be named 'artist'.
    - The second column refers to the name of the song and must be named 'song'.
    - The third column refers to the release year of the song and must be named 'year'.
    - The fourth column refers to the column used to sort the data and must be named 'popularity', 'energy', or 'danceability'.