Assignment 3

Due: Tuesday, March 18, 2025 (11:59 pm)

Submission via Git only

Overview

Programming environment

For this assignment, you must ensure that your code executes correctly on the UGLS server, which you learned about as part of Lab 01. This same environment will also be used by the teaching team to evaluate your submitted work. You must ensure that your program compiles, links, and executes perfectly on UGLS. **If your program does not run on UGLS, even if it works fine on your local computer, your submission will receive 0 marks**. All test files and sample code for this assignment are available on your Git repository. After cloning your repository, you can use the following command to obtain the sample code (inside your repository, of course):

```
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/

Objectives of this assignment

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 lose your work.

IV. Test your code against the provided test cases.

This assignment: spf_analyzer, using C's heap memory

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

- During each execution, your program must complete one of the specified tasks and generate a corresponding CSV file named output.csv. To assist with implementation, the expected solution for each task (i.e., the CSV file your program should produce) is provided in the tests folder within the a3 directory of your Git repository. The following table lists the expected tasks to be supported by your implementation and the expected outputs.

TABLE 1.[1]

Task	Expected Output
1. Generate a CSV file containing the Record_ID and Exam_Score for students who meet the following conditions: Attendance is exactly 100%, and Extracurricular_Activities is marked as 'Yes'. Sort the results by Record_ID in ascending order. Only 20 records should be displayed.	a3/tests/test01.csv
2. Generate a CSV file containing the Record_ID, Hours_Studied and Exam_Score for students who studied more than 40 hours. Sort the results by Exam_Score in descending order. Only 10 records should be displayed.	a3/tests/test02.csv
3. Generate a CSV file containing the columns Record_ID, Hours_Studied, and Exam_Score, but only for entries where Exam_Score is at least 85. Sort the results primarily by Exam_Score in ascending order; if two scores are identical, sort by Record_ID in ascending order. Only 10 records should be displayed.	a3/tests/test03.csv

[1] An outlier is a data point that differs significantly from others. In the provided date, a student scored 101 on the exam, exceeding the normal 0-100 range.

- You need to allocate storage on the heap dynamically to solve this assignment. Dynamic data structures must be used in the form linked-list manipulation routines (i.e., using only arrays is not permitted for this assignment).
- The program itself consists of several C source files and a makefile for build management:
    a. emalloc[.c or .h]: Code for safe calls to malloc, as is described in labs\lectures, is available here.
    b. list[.c or .h]: Type definitions, prototypes, and codes for the singly-linked list implementation described in labs/lectures.
    c. makefile: This automates many of the steps required to build the spf_analyzer executable, regardless of what files (.c or .h) are modified. This file can be invoked using the Bash command make.
    d. spf_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 as the only collection in your program. You must use linked lists instead. The elements of the linked list can be struct types.**

Testing your solution

- Similarly to A2, this time the tester file will execute your program, but it won't compile it (you need to use make to compile your code). You'll only need to pass the number of the test 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 tasks. 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., spf_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. **If you modify the makefile, it is your responsibility to make sure that it will run in UGLS.**

Input specification

1. The input dataset for this assignment is a simplified version of the Student Performance Factors dataset from Kaggle.com.
2. Refer to https://www.kaggle.com/datasets/lainguyn123/student-performance-factors for more information about each column/property for each record (e.g., Hours_Studied).

Output specification

1. All output must be generated in a file called *output.csv*.
2. Refer to the tests folder for more information about the expected output for each test.

Evaluation

We'll use the same evaluation rubric from A1.