Assignment 1

Due: Thursday, February 6, 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

- Understand a problem description, along with the role of the provided sample input and output.
- Extend the functionality of a C program.
- Use the programming language C to implement a Unix filter—a text processor named *spf_analyzer.c*—**without resorting to dynamic memory allocation.**
- Use *git* to manage changes in your source code and annotate the continuous evolution of your solution with "messages" given to commits.
- Test your code against the provided test cases.

This assignment: *spf_analyzer.c*

In this assignment, you will learn C by solving a problem involving the CSV and YAML file formats, which are widely used in many areas of computing. Files following the CSV specification typically have the ".csv" filename suffix, while files following the YAML specification usually have the ".yml" or ".yaml" filename suffix.

The primary objective of this assignment is to integrate and process data from two distinct input files (i.e., datasets), each containing information about curricular and extracurricular factors influencing student performance. The combined data will be used to address typical data exploration tasks commonly found in data science projects. This assignment focuses on analyzing factors such as sleep hours, attendance, tutoring sessions, and other variables that may affect a student's final exam score.

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 a1 directory of your Git repository.

The following table lists the expected tasks to be supported by your implementation and the expected outputs.

TABLE 1.

Task	Expected Output
1. Generate a CSV file containing the Record_ID and Exam_Score for students who scored above 90 on the final exam.	a1/tests/test01.csv
2. Generate a CSV file that contains all records (with their respective properties) from the extracurricular dataset.	a1/tests/test02.csv
3. Generate a CSV file with all rows and columns from the merged dataset (i.e., merge between a1-data-curricular.csv and a1-data-extracurricular.yaml) where the Exam_Score exceeds 90.	a1/tests/test03.csv
4. Generate a CSV file containing the Record_ID and Exam_Score for students who had 100% attendance.	a1/tests/test04.csv
5. Generate a CSV file containing the Record_ID and Exam_Score for students whose Sleep_Hours were greater than or equal to their Hours_Studied.	a1/tests/test05.csv
6. Generate a CSV file that includes the Record_ID, Exam_Score, and Extracurricular_Activities for students who scored below 60 on the final exam.	a1/tests/test06.csv

Exercises for this assignment

To streamline the development of spf_analyzer.c, consider breaking the problem into smaller, more manageable components. To support this approach, we recommend dividing the assignment into the following parts:

Part 1: Understand the provided code

- An initial implementation of spf_analyzer.c is provided. This initial implementation reads the a1-data-curricular.csv dataset , uses an array of C structs to store the content of the dataset in memory, and prints out the content of the read file to standard ouput.

- You will need to modify the initial implementation to meet the requirements of the assignment. Feel free to reuse any components from the provided implementation as needed.
- Consequently, studying and understanding the provided code is crucial to complete this assignment.

Part 2: Argument processing

- The first step in adapting your solution to complete the assignment is ensuring that spf_analyzer.c can accept and process an argument specifying the task to be performed during program execution. Specifically, when the program is run (as will be automated and explained later in this document), it will be executed using a command in the following format:

```
./spf_analyzer --TASK="3"
```

- Ensure that the program retrieves the TASK argument from the command line using the char *argv[] parameter in your main function.
- To get the actual value of the argument, we recommend using the function sscanf(), although strtok() is also a valid alternative.

Part 3: Read dataset content

- Your program must read the datasets required by spf_analyzer.c line by line.
- The code for reading the first dataset (a1-data-curricular.csv) is already provided.
- You need to add additional code (preferably encapsulated in functions) to read the second dataset (a1-data-extracurricular.yaml). Note that the second dataset uses a different format (YAML) compared to the first (CSV), so you will need to implement a method to handle this format.
- You don't need extensive knowledge of YAML for this assignment. It is sufficient to understand that each record (analogous to a row in the CSV file) contains a set of properties/attributes(e.g., Extracurricular_Activities, Physical_Activity, Record_ID, Sleep_Hours). Records in the YAML file are separated by a hyphen (-).
- Both datasets share a common property, Record_ID, which serves as a key to match records between the two files. A complete record for a student requires combining information from both datasets using this shared Record_ID.
- The approach to processing the YAML file should be similar to the one provided for the CSV dataset. Specifically:
    1. Open the file (e.g., using fopen()).
    2. Parse the file line by line (e.g., using fgets() with for or while loops).
    3. Store the fields of each record in an array of structs.
- Ensure your program always reads the datasets from the same fixed file paths when executed:
    1. data/a1-data-curricular.csv
    2. data/a1-data-extracurricular.yaml

Part 4: Process lines

- After your program successfully reads all the lines from both input files and stores the data in memory using an appropriate structure (e.g., structs), you will need to augment your code with additional functions to implement the tasks outlined in **TABLE 1**.
  • Keep in mind that each execution of your program is designed to solve a specific task, as specified by the argument passed to the program.

Part 5: Export output

- After each execution, your program must generate a file called output.csv containing the desired ouput for the task passed as an argument to the program.
- Use the fopen() function and format specifiers to generate the required output.

Part 6: Test the output of your program against the provided tester file

- Compile your program using the -std=c99 flag. That is, before testing your program, you must execute the following command:

```
gcc spf_analyzer.c -std=c99 -lm -o spf_analyzer
```

- In order check for correctness, **please use the provided *tester* file**. The *TESTS.md* markdown file describes the usage of this program for each of the tests. This is method that will be used by the instructors when grading your assignment.
- For example, if you plan to check that your program passes the first test, you must execute the following command

```
./tester 1
```

- **It is very import to use the *tester* file for testing. Your output must exactly match the expected test output for a test to pass. Do not attempt to visually check test cases — the eye is often willing to deceive the brain, especially with respect to the presence (or absence) of horizontal and vertical spaces. Please note that *tester* will indicate that a test failed if there is extra or missing white space (or extra empty lines at the end of the file).**

Requirements and recommendations

1. **You MUST use the -std=c99 flag when compiling your program as this will be used during assignment evaluation (i.e., the flag ensures the 1999 C standard is used during compilation).**
2. **DO NOT use malloc(), calloc() or any of the dynamic memory functions.**
3. Keep all of your code in one file for this assignment (that is, *spf_analyzer.c*). In later assignments we will use the separable compilation facility available in C.
4. Use the test files to guide your implementation effort. Start with the simple test 01 and move onto 02, 03, etc. in order. **Refrain from writing the program all at once, and budget time to anticipate when things go wrong!**

5. For this assignment you can assume all test inputs will be well-formed (i.e., the teaching team will not evaluate your submission for handling of input or for arguments containing errors). Later assignments might specify error-handling as part of their requirements.
6. Use git when working on your assignment. Remember, that the ONLY acceptable method of submission is through Git.

What you must submit

A single C source file named **spf_analyzer.c**, submitted to the a1 folder **in your Git repository**. Git is the **only** acceptable way of submission.

Input specification

1. The input datasets for this assignment are 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.