Assignment 1

Due: Wednesday, February 8, 2023 (11:59 pm)

Submission via Git only

Overview

Programming environment

For this assignment you must ensure your code executes correctly on the virtual machine or the programming environment (for M1\M2 computers) you configured as part of Lab 01. This is our "reference platform". 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 compile, link, and execute 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 on your Git repository. Git will be discussed in detail during the week of January 30th. Nevertheless, you can clone your repository by running the following command (only needs to be executed once):

```
git clone ssh://NETLINKID@git.seng.uvic.ca/seng265/NETLINKID
```

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.
- Use the programming language C to implement a Unix filter—a text processor named *route_manager.c*—without resorting to dynamic memory allocation.
- Leverage Unix commands, such as diff, in your coding and testing. The lab instructors will explain in the labs how to use the *diff* command.
- 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: *route_manager.c*

In this assignment, you will learn C by solving a problem involving the *CSV* file format that is widely used in many aspects of computing. Files following the *CSV* specification usually have the ".csv" filename suffix.

Once you get the required files for Assignment 1 from your repository, you will find the main *airline-routes-data.csv* input file[1]. Except for the first one (i.e., the name of the fields\columns), each line in this file represents an airline route, containing the following fields:

Field Name	*Description*
airline_name	The name of the airline offering the route.
airline_icao_unique_code	Unique identifier of the airline offering the route.
airline_country	Registered location (country) of the airline offering the route.
from_airport_name	Name of the source airport for the route.
from_airport_city	City where *from_airport_name* is located.
from_airport_country	Country where *from_airport_city* is located.
from_airport_icao_unique_code	Unique ICAO identifier for *from_airport_name*.
from_airport_altitude	Registered altitude for *from_airport_name*.
to_airport_name	Name of the destination airport for the route. .
to_airport_city	City where *to_airport_name* is located.
to_airport_country	Country where *to_airport_city* is located
to_airport_icao_unique_code	Unique ICAO identifier for *to_airport_name*.
to_airport_altitude	Registered altitude for *to_airport_name*.

[1] This file was generated from https://www.kaggle.com/datasets/arbazmohammad/world-airports-and-airlines-datasets

You are to write a C program that inputs data from a provided .csv file, accepts options and arguments from the command line, and then outputs to the console airline routes based on user-defined filters. To get started, the teaching team has provided a skeleton version of *route_manager.c*.

Depending on filters (i.e., arguments) provided by the user, *route_manager.c* will generate a list with the airline routes that meet the conditions established by the filters. Suppose you type the following arguments into your program (i.e., Test 3):

```
./route_manager --DATA="airline-routes-data.csv" --AIRLINE="ACA" --DEST_COUNTRY="Bahamas"
```

Then the output to be produced is as follows:

```
FLIGHTS TO Bahamas BY Air Canada (ACA):
FROM: CYUL, Montreal, Canada TO: San Salvador (MYSM), Cockburn Town
FROM: CYYZ, Toronto, Canada TO: Exuma Intl (MYEF), Great Exuma
FROM: CYYZ, Toronto, Canada TO: Lynden Pindling Intl (MYNN), Nassau
```

In case there are no routes meeting the provided criteria, the program's output should be as follows (e.g., Test 1):

```
./route_manager --DATA="airline-routes-data.csv" --AIRLINE="SWR" --DEST_COUNTRY="Argentina"
```

Then the output to be produced is as follows:

```
NO RESULTS FOUND.
```

At the end of this document is a more detailed specification for the input filter your program must read, parse, and process, as well as syntax and format expected in the output. Since such specifications frequently are ambiguous, the provided test-output files (i.e., files that begin with the letters *"test"*) represent the required output format. The *TESTS.md* markdown file describes each of the nine tests, with corresponding test outputs listed as *test01.txt, test02.txt,* etc.

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.

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).

Exercises for this assignment

To facilitate the development of route_*manager*, try to decompose the problem into small and more manageable parts. To promote this, we suggest dividing the assignment into the following parts.

Part 1: Argument processing

- Obtain the arguments from the command line using the char *argv[] parameter of your main function.
- To get the actual value of the argument, we recommend using the function sscanf() (although you can also use strtok()).

Part 2: Read file content

- You will need to read line by line the file passed as an argument to *route_manager*. The functions fopen() and fgets() together with for or while loops can readily accomplish this task.

Part 3: Process lines

- Once your program can read all the lines from the input *.csv* file, you will need to process each line to obtain relevant data to produce the required output.
- You can use the function strtok() to obtain data from the fields from each route.
- Another option is to create some representation of the routes in the file using a struct (optional but recommended).

Part 4: Export output

- Use the fopen() function and format specifiers to generate the required output in a text-based file called **output.txt**.

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, *route_manager.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 example in 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!** Use the Unix command *diff* to compare your output with what is expected.
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 **route_manager.c**, submitted to the a1 folder **in your Git repository**. Git is the **only** acceptable way of submission.

Evaluation

Assignment 1 grading scheme is as follows.

A grade: A submission completing the requirements of the assignment which is well-structured and clearly written. Global variables are not used. route_manager runs without any problems; that is, all tests pass and therefore no extraneous output is produced. This is a good submission that passes all the tests and does not have any overall 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). A solution with minor issues will be given an A- (80-84 marks).

B grade: A submission completing the requirements of the assignment. route_manager 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 compiles 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 compiles with warnings
- Submission has 1 or 2 large functions.
- Program or file-scope variables

C grade: A submission completing most of the requirements of the assignment. route_manager runs with some problems. This is a submission that present 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). route_manager 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 compile, 0 marks. Submissions that do not run, 0 marks. Submissions that fail all tests and show a very poor to no effort (as evaluated by the marker) are given 0 marks. Submissions that fail all tests, but represent a sincere effort (as evaluated by the marker) may be given a few marks.

In general, straying from the assignment requirements will be penalized severely.

Additional Criteria for Qualitative Assessment

IMPORTANT: Only submissions that satisfy **ALL** the criteria described below will be considered for an A+ grade.

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 overall quality assessment.

Functional decomposition: quality coding requires the good use of functions. Code that relies on few large functions to accomplish its goals is very poor-quality code. Typically, a good program has a main function that does some basic tasks and calls other functions, that do most of the work. A solution that passes all tests, but contains all code in one or two large functions will not be given a grade better than a B. You must not use program-scope or file-scope variables.

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.

Debugging / Comment artifacts: You must submit a clean file with no residual commented lines of code or unintended text.

Quality of solution: the 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 which represent the data in inappropriate data structures, solutions which scale unreasonably with the size of the input.

Input specification

1. All input is from airline-routes-data.csv.
2. Three different combinations (i.e., use cases) of filters\arguments can be provided to the program (notice the references to the fields from the previous table):

Combination Name	Filters\Arguments		Example
	Name	Description	
Use Case 1	DATA	Path to airline-routes-data.csv	--DATA="airline-routes-data.csv" --AIRLINE="ACA" --DEST_COUNTRY="Bahamas"
	AIRLINE	Possible value for airline_icao_unique_code	
	DEST_COUNTRY	Possible value for to_airport_country	
Use Case 2	DATA	Path to airline-routes-data.csv	--DATA="airline-routes-data.csv" --SRC_COUNTRY="India" --DEST_CITY="Tokyo" --DEST_COUNTRY="Japan"
	SRC_COUNTRY	Possible value for from_airport_country	
	DEST_CITY	Possible value for to_airport_city	
	DEST_COUNTRY	Possible value for to_airport_country	
Use Case 3	DATA	Path to airline-routes-data.csv	--DATA="airline-routes-data.csv" --SRC_CITY="Toronto" --SRC_COUNTRY="Canada" --DEST_CITY="Cancun" --DEST_COUNTRY="Mexico"
	SRC_CITY	Possible value for from_airport_city	
	SRC_COUNTRY	Possible value for from_airport_country	
	DEST_CITY	Possible value for to_airport_city	
	DEST_COUNTRY	Possible value for to_airport_country	

Output specification

1. All output must be generated in a file called *output.txt*.
2. Depending on the use case (i.e., combination of filters\arguments) provided the output should be as follows (notice the references to the filters\arguments and fields from the previous tables):

Use Case	Expected Output	Example
Use Case 1	FLIGHTS TO DEST_COUNTRY BY airline_name (AIRLINE): FROM: from_airport_icao_unique_code, from_airport_city, from_airport_country TO: to_airport_name (to_airport_icao_unique_code), to_airport_city	test03.txt
Use Case 2	FLIGHTS FROM SRC_COUNTRY TO DEST_CITY, DEST_COUNTRY: AIRLINE: airline_name (airline_icao_unique_code) ORIGIN: from_airport_name (from_airport_icao_unique_code), from_airport_city	test05.txt
Use Case 3	FLIGHTS FROM SRC_CITY, SRC_COUNTRY TO DEST_CITY, DEST_COUNTRY: AIRLINE: airline_name (airline_icao_unique_code) ROUTE: from_airport_icao_unique_code-to_airport_icao_unique_code	test08.txt