Assignment 3

Due date: Wednesday, March 22, 2023
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 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
```

Hint: To verify whether you uploaded the files properly, 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). 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

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: route_manager.c, using C's heap memory

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

a) Based on provided arguments and data file (i.e., dataset), route_manager will help us answer the first three questions from A2:

q1: What are the top N airlines that offer the greatest number of routes with destination country as Canada?

q2: What are the top N countries with least appearances as destination country on the routes data?

q3: What are the top N destination airports?

However, this time N is an input provided by the user.

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).

c) 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**).

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.
- **list[.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 route_manager executable, regardless of what files (.c or .h) are modified. This file can be invoked using the Bash command make.
- **route_manager.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

- 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 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., route_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. If you modify the makefile, it is your responsibility to make sure that it will run in the reference platform (i.e., VM).

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. route_manager 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. 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 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. route_manager 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). 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 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 is in one single file in the .YAML format (i.e., routes-airlines-airports.yaml) and come from the "Worlds Airports and Airlines Datasets" dataset in Kaggle.com.
- YAML is a simple data-representation format that is widely used across the software engineering and data science lifecycles. We only need basic concepts from the YAML format for this assignment. You can refer to online resources (e.g., this 5-min video) for a brief overview of the language.
- Please refer to the A1's write up or the original data source for a description of the fields in the input data files.

Program Arguments

Argument	Description	Example
--DATA	Location of the routes-airlines-airports.yaml file.	`./route_manager --DATA="routes-airlines-airports.yaml" --QUESTION=1 --N=10`
--QUESTION	Question (e.g., **q1**, **q2**) to be answered by the program.	
--N	The number of elements to be displayed from the answer to the question.	

Output specification

- After execution, your program must produce\create the following file:
  - **A. output.csv**: This file will contain two-dimensional data (i.e., a table with two columns) that represents the answer to the question passed as an argument to the program (e.g., **q1**, **q2**, **q3**) and the displays only the number of elements defined by the user (i.e., N).
    - The first column in the table (i.e., X axis) must be named 'subject'.
    - The second column in the table (i.e., Y axis) must be named 'statistic'.
    - Examples of the expected output.csv files for each test are provided (i.e., tests\test01.csv, tests\test02.csv, etc.).