Data Processing from CSV files Using Linked Lists

Implement a C program, **topcgpas.c** that can read a CSV file containing student information and print a list of top 5 students (based on their CGPAs) by using a linked list to process this data.

A sample format of the input CSV file is as shown below

```
sid,email,lname,fname,cgpa
2123123513,joack@quendeldon.edu,Moe,Jack,3.1
2173413424,leashell@quendeldon.edu,Shell,Leache,3.0
```

The first line is the header (names of fields) followed by actual information. As can be seen, the student ids are 10 digit numbers, followed by email, last name, first name and the CGPA of the student, X.Y a floating point number.

The data records themselves are not in any particular order (random).

1. Use **vim** to create a C program source file **topcgpas.c**. All of your source code logic should be contained in this file. You may write multiple functions, etc. (You are encouraged to do so). **Please read through the entire document before you start working, as the programming constructs that you are allowed to use are limited in some cases.**
2. The source code file should include a comment section at the beginning that describes its purpose (couple of lines), the author (your name), your department, a small "history" section indicating what changes you did on what date.

```
/*
 Program to generate a report on the students having the top CGPAs
 ************************************************************
 * Author         Dept.             Date          Notes
 ************************************************************
 * Joseph D       Comp. Science. Mar 20 2021    Initial version.
 * Joseph D       Comp. Science. Mar 21 2021    Added error handling.
 * Joseph D       Comp. Science. Mar 24 2021    Fixed bug in output logic.
*/
```

   The code should be properly indented for readability as well as contain any additional comments required to understand the program logic.

3. The source code should be compilable by **exactly** using the following command:
   gcc -o topcgpas topcgpas.c

4. Compilation should not produce any warnings!

5. The program accepts 2 arguments, the first is the name of the input CSV file, and the second is the name of the output CSV file. These file names could be anything, and may have absolute or relative paths (see below).

```
$ ./topcgpas students.csv output.csv
```

6. If the program is not passed exactly 2 arguments, it should print a usage message and terminate with code 1.

```
$ ./topcgpas
Usage ./topcgpas <sourcecsv> <outputcsv>
$ echo $?
1
$
```

7. If the program cannot open the input file, it should display an error message and terminate with code 1.

```
$ ./topcgpas nosuch.csv output.csv
Error! Unable to open the input file nosuch.csv
$ echo $?
1
$
```

8. If the input file is empty, the program should display an error message and terminate with code 1.

```
$ ls -l empty.csv
-rw------- 1 jdsilv2 root 0 Mar 20 15:54 empty.csv
$ ./topcgpas empty.csv output.csv
Error! Input CSV file empty.csv is empty
$ echo $?
1
$
```

9. The program should behave the same as above, if the input file only has the header (as from the program's perspective there is no data for it to work on).

```
$ cat headeronly.csv
sid,email,lname,fname,cgpa
$ ./topcgpas headeronly.csv output.csv
Error! Input CSV file headeronly.csv is empty
$ echo $?
1
$
```

10. The program should write the student ID (sid), email and cgpa of the top 5 (based on their CGPAs) students into the output file. The output file should have a header and the records should be printed with the students with the highest CGPAs on the top (i.e., ordered based on decreasing CGPAs). If two students have the same CGPA, they can be in any order. Existing output files should be overwritten (not appended). On successful execution, the program should terminate with code 0. (see next page)

```
$ cat students1.csv
sid,email,lname,fname,cgpa
2123123513,joack@quendeldon.edu,Moe,Jack,3.1
... truncated for brevity ...
2161242311,pcruz@quendeldon.edu,Cruz,Pedro,2.5
2161242311,alope@quendeldon.edu,Lopez,Angela,3.5
$ ./topcgpas students1.csv output.csv
$ echo $?
0
$ cat output.csv
sid,email,cgpa
2161242311,alope@quendeldon.edu,3.5
2169133134,hholmes@quendeldon.edu,3.4
2173413424,leashell@quendeldon.edu,3.3
2178423131,clairec@quendeldon.edu,3.1
2123123513,joack@quendeldon.edu,3.1
$
```

11. If the program was not able to open an output file for writing, it should display an error message and terminate with code 1.

```
$ ./topcgpas students1.csv cannot.csv
Error! Unable to open the output file cannot.csv
$ echo $?
1
$
```

12. The program should create the output file **ONLY** if it is going to produce valid output.

13. If the input file has less than 5 students, they should still make it to the output, in the proper format and order as described above.

14. If there are any other students with the same CPGA as the fifth student that was written to the output, all those additional students must also be written to the output file (i.e., the total number of students written to the output file can go above five in this particular case).

15. **[VERY IMPORTANT]** Your program must read through the input CSV file **ONLY ONCE**, adding a student record in the linked list as you read through each student data in the input CSV file. After that, your program logic should then process this linked list to find the top CGPA students. You should not copy over the student / CPGA information into a new array, etc. You can use the below structure as the "node" of the linked list.

```
struct StudentRecord
{
  long sid;
  char email[30];
  char lname[20];
  char fname[20];
  float cgpa;
  struct StudentRecord *next;
};
```

16. If the program runs out of memory when using **malloc** or **calloc**, it should display an error message and terminate with code 1.

```
$ ./topcgpas students1.csv output.csv
Error! program ran out of memory
$ echo $?
1
$
```

17. **[VERY IMPORTANT]** The program must be completely written in C, and should not involve shell scripts, commands, etc. (for example invoked using the **system** function call.)

18. You are **ONLY** allowed to use the library functions from the libraries **stdio.h stdlib.h**, and **string.h**. Feel free to write your own functions.

19. The format of the output file should match what is given in the instructions above.

20. Ensure the program does not crash for any test cases.

21. If you program allocates memory using **malloc** or **calloc**, make sure that you free all of them explicitly at the end of the program (even if it terminates without producing an output).

22. Ensure that the program does not get stuck in an infinite loop when it is tested.

Assumptions:

- You can assume that the values of different fields of student records will fit within the individual members of the structure given as part of the assignment description. This should help you understand how "wide" a record can get and then you can make use of that information in reading the input CSV file.

- You do not have to look for "bad data" in the student records - i.e, no student ids with non-numeric characters in it, etc.

- You can assume that the names of the fields in the header and their order remains the same.

Some hints to get started:

Here are some helpful recommendations (following these hints is not strictly necessary)

- Read a line at a time using **fgets** into a character array of sufficient size.

- Use the function **strtok** to parse the individual fields. However, I highly recommend that you read this article (https://icarus.cs.weber.edu/~dab/cs1410/textbook/8.Strings/strtok.html) on how **strtok** works internally before you venture out.

- Once a student record is created, add it to the linked list. You might find it beneficial for the rest of your program logic to add any new nodes in such a way that the list maintains the decreasing order of CGPAs.

- Once you have read all the student information into the linked list, traverse the linked list and write the required information to the output file. You will have to use some logic to keep track of as to when to stop writing.

Restrictions:

- Please make sure to read and follow the instructions given in this document and only include features and functionality that you are allowed to use.

Testing:

- You are encouraged to write tester scripts to test your program's behaviour. Start with the very basic (like no arguments, empty file, file that has only a header, file with couple of records, etc.) so that you can easily look at the output to see if the basic logic is working before getting into more complex test cases.