CX 4010 / CSE 6010
Assignment 4
Flight Delays
Additional requirements for graduate students only

Initial Submission Due Date: 11:59pm on **Thursday, October 27**
Peer Reviews Due Date: 11:59pm on **Thursday, November 3**
Final Submission Due Date: 11:59pm on **Thursday, November 10**
Submit codes as a single zipfile as described herein to Canvas
Submit peer reviews through Canvas
48-hour grace period applies to all deadlines

In this assignment, you will write a program to calculate and output information about flight delays at US airports. Essentially, you will need to perform the following tasks.

- Read in flight information (airline, flight number, departure and arrival airport codes, scheduled and actual departure and arrival times) from a file specified as a command-line argument.
- Calculate the arrival delay (if any) in minutes for each flight. (Flights that arrive before or at the scheduled arrival time should have a delay of 0.)
- Accumulate the arrival delay information according to the arrival airport.
- Sort the airports according to average arrival delay (averaged over all arriving flights) in descending order, with any ties broken by sorting in alphabetical order by the three-letter airport code.
- Write the sorted output to a file, including for each airport its average arrival delay over all arriving flights, the average arrival delay over all flights with a delay, its total number of arriving flights, and its total number of delayed arriving flights.

Input File Information and Format: Your program will read in an input file of flight information for a single day and should be able to work for file of unspecified length (that is, the number of flights, and hence the number of lines in the file, is not known in advance). The first line will provide the number of arrival airports as an integer.

After the first line, each line in the file consists of eight fields separated by commas. Here is an example.

```
DL,2561,ABE,ATL,600,610,818,815
```

The six fields, from left to right, have the following explanations.

- Two-letter airline code
- Flight number (integer, maximum of 4 digits)
- Three-letter code for departure airport
- Three-letter code for arrival airport
- Scheduled departure time, given in a modified “military style” (no colon to separate hours and minutes, times ranging from 0 to 2359 without am/pm designations, no leading zeros)
- Actual departure time (same style as for scheduled departure time)
- Scheduled arrival time (same style as for scheduled departure time)
- Actual arrival time (same style as for scheduled departure time)

You are provided with a sample input file (test_times_new.csv) with matched output file that you can use to test whether your program works. You are also provided with a larger input file (flights_jan1_delta_times_reg.csv) with no matched output file for further testing. Note that your program will be tested with a different but similar file. The input file name should be specified as the first command-line argument.

Storing information: For this assignment, you should store information about each airport using an array of structs. You should define the struct as you see fit. The array **must be allocated dynamically**, using as its dimension the integer specified in the first line of the input file. You may define any other structs as you see fit.

Calculating delays: For each flight, the arrival delay should be calculated in minutes as the difference between the actual arrival time and the scheduled arrival time. If the flight is on time or early, the delay should be set to 0. Be sure your program correctly calculates the delay in minutes. Note that because the file does not include date information, you will need to be careful calculating delays when flight arrivals are around midnight. For example, if the scheduled arrival time is 2355 and the actual arrival time is 5 (0005), this flight should be considered as arriving 10 minutes late, rather 23 hours and 50 minutes early. Conversely, if the scheduled arrival time is 5 (0005) and the actual arrival time is 2355, this flight should be considered as arriving 10 minutes early, rather than 23 hours and 50 minutes early. For convenience, you may assume that no flight is more than 2 hours early and no flight is more than 23 hours late.

Sorting methods: Undergraduate students should implement two sorting methods: insertion sort and another merge sort. Graduate students should implement three sorting methods: insertion sort, heap sort, and merge sort. User selections for sorting should be specified by a second command-line argument (the first argument must be the input file name). Use arguments of "i" for insertion sort, "m" for merge sort, and (for graduate students) "h" for heap sort. The default case (no command-line argument) should use insertion sort. **For the initial submission, implementation of only one sorting method is fine.** For the output file, sort the airports according to the average arrival delay (sum the delays for that airport and divide by the total number of flights that arrived at that airport) in ***descending*** order (that is, the airport with the largest delay should be listed first and airports without delays last). When more than one airport has the same average arrival delay, your output should list those airports in alphabetical order by the three-letter airport code. Although you are required to use the sorting method specified on the command line when sorting by average arrival delay, you may use any method you like when you must sort by airport code.

Output file (note that the output file should be the same regardless of the sorting algorithm used):

Your output file should be named **sortedoutput.txt**. You are provided with an output file corresponding to a sample input file, and your file should produce an identical output file, which you can check with a variety of tools, such as the diff utility in linux.

The first row of the output file should include column headings as shown in the example. You may use the following statement (modified to print to your output file); the format string has the correct text including spaces to match the provided output file.

```
printf("Airport Abbrev   Delay-All   Avg Delay-Late   #Arrivals
#Late Arrivals\n");
```

Each remaining line should list airport information in sorted order, where the airport information consists of the three-letter airport code, the average arrival delay over all flights arriving to that airport, the average arrival delay over all delayed flights arriving to that airport, the total number of flights arriving to that airport, and the total number of delayed flights arriving to that airport. You may use the following statement (modified to print to your output file and to print the appropriate variables).

```
printf("      %s %15.3f %15.3f %11d %11d\n",var1,var2,var3,var4,var5);
```

Apart from the requirements indicated above, you may organize your program any way you like, including the use of any supporting .h and .c files. Your main program should be named **airportmain.c**.

Your code (final submission) should successfully compile (using gcc) and run on the COC-ICE cluster. This does not mean we necessarily will try to run your code on the cluster, but if we have any difficulties compiling and running it, we will move it to that environment for further testing. If you are compiling and running your codes under linux, most likely you will not need to make any changes as a result of this requirement.

Hints for reading in datasets that include strings:

- You can put commas in the format string.
- You can read strings using %s in the format string and can limit length using, e.g., %3s.
- When allocating space for a string variable, include enough space to accommodate one extra character (the "end of string" character). Without it, results may be unpredictable. For example, a 3-letter string would require declaration of a string variable along the lines of `char mystring[4]`. However, reading in the 3-letter string would only need %3s in the format string (not %4s).
- For this assignment, you may find it useful to determine the end of file by checking whether `fscanf()` returns the expected number of items. The return value of `fscanf()` indicates the number of items read. When the end of the file is reached, `fscanf()` will be unable to read in the expected number of items.

Useful functions for working with strings in this assginment:

- The function `strcpy(destination,source)` is a good choice for copying a source string to another destination (the standard assignment operator = will not work for strings). `strcpy(destination,source)` will copy the string pointed to by `source` to the location pointed to by `destination`. Because a string is an array of chars, you can use the names of your strings directly.
- The function `strcmp(string1,string2)` is a good choice for comparing strings (standard operators for comparing the values of numbers like > and == will not work for strings). The return value is 0 if the strings are the same and is signed to indicate precedence for strings that are not the same (e.g., a negative value indicates the first string argument precedes the second string argument).
- You will need to include `string.h` to use these string-related functions.

Management:

- Don't forget to free any memory allocated dynamically!
- Don't forget to close any input/output files you open!

You should submit to Canvas a single zipfile that is named according to your Georgia Tech login—the part that precedes @gatech.edu in your GT email address. To receive full credit, your code must be well structured and documented so that it is easy to understand. Be sure to include comments that explain your code statements and structure.

The zipfile should include the following files:

(1) your code (all .c and .h files). If you are using linux or Mac OS, you may use a makefile to compile and run your program, and you should include it if so. Do not include any input or output files.

(2) a README text file (not formatted in a word processor, for example) that includes the compiler and operating system you used for compiling and running your code along with instructions on how to compile and run your program.

(3) a series of 3 slides composed in PowerPoint or similar software, saved either in PowerPoint or as a PDF and named slides.pptx or slides.pdf, structured as follows:

- Slide 1: your name and a brief explanation of how you developed/structured your program (data structures, functions, program flow, etc.). You are limited to one slide.
- Slide 2: brief comments on the performance of the different sorting methods and a brief statement explaining why you believe your program works correctly. You are again limited to one slide; focus on what you think is most important.
- Slide 3: a brief exposition of what you found (or did not find) useful about the peer review process.

Initial submission

Your initial submission does not need to include the slides. It will not be graded for correctness or even tested (including for compatibility with gcc on the COC-ICE cluster) but rather will be graded based on the appearance of a good-faith effort to complete the majority of the assignment. However, you should be aware that your peers will review your initial submission. Thus, it is recommended you complete as much of the main functionality of the assignment as possible, even if you are continuing to adjust certain aspects (e.g., command-line arguments, writing to a file rather than to the screen, additional sorting algorithms, etc.). It is also recommended that you include your README file.

Peer review assignment

In this assignment, you will have two weeks between the initial and final submission to allow you the opportunity to review others' submissions and provide feedback. Although the feedback you receive from your peers should be useful, we also expect that having the opportunity to view others' code will be a valuable experience for you as well. You will submit your final submission after the peer reviews have been completed so that you can benefit from the process. However, while you may find yourself inspired by what others have done, you should be careful not to plagiarize from others' code. You should write and submit your assigned peer reviews while taking a break from looking at your code and you should not look at others' code again afterward.

Peer review assignments will be made in Canvas after all initial assignments have been submitted. (peer review assignments should be made by Monday, October 10). If you do not submit an initial assignment on time, you may not be able to complete the peer review assignment either.

You will be assigned three peer reviews. Your peer review does not have to be long but it should address the following 5 topics. Use the Comment box to submit your peer review.

- Program execution: does the program compile and execute without error, and is the output clear?
- Program structure and implementation: is the structure of the code clear and easy to follow and are the implementation choices good?
- Program style and documentation: are the comments in the program useful and at the appropriate level, is the program easy to follow visually through indenting and blank lines, and does the program avoid copying the same code in multiple places (redundancy)?
- Overall strengths of the code.
- Any constructive suggestions for improvement.

Because of Canvas limitations, grades for the peer reviews you write will be reported in a separate peer review assignment. (You will not need to do anything extra for the official Canvas assignment, but that is how the grades will appear.) Peer reviews you receive will not affect your grade in any way. For information on how to use Peer Reviews in Canvas, including where to submit your peer reviews and where to find peer reviews of your submission, please see the following videos (or other similar ones you may find). **In addition to (not instead of!) submitting your reviews as comments so that your peers can see them, please also copy and paste your review text and submit as a text input or upload a file with the text of your three peer reviews in the separate Peer Review assignment.**

https://www.youtube.com/watch?v=cKyvKKq0mtc
https://community.canvaslms.com/t5/Video-Guide/Feedback-Overview-Students/ta-p/383514