School of Computing and Information Systems

comp20005 Intro. to Numerical Computation in C
Semester 1, 2024
Assignment 2

Learning Outcomes

In this project you will demonstrate your understanding of array of structs, and the engineering processes used to assemble and test a non-trivial program.

Background and Task

Scientific, engineering, and financial datasets are often stored in text files using *comma separated values* (`.csv` format) or *tab separated values* (`.txt` or `.tsv` format), usually with a header line describing the contents of the columns. The simplest framework for processing such data is to first read the complete set of input rows into arrays, one array per column of data to be manipulated, and then pass those arrays (and a single buddy variable) into functions that carry out the required data transformations and analysis, knowing exactly what the data represents.

In this approach a separate program is then needed for each type of input data. Much better is to have a generic program that is capable of handling *all* types of CSV data. And that's what you are going to work on in this assignment.

Each input file will start with a header line that gives symbolic names to the columns of data values stored in the file. That line will contain anywhere between 1 and 50 words, one word per column, separated by commas. An example appears below. A set of as many as 10,000 data lines then follows. Each data line contains nothing but numbers, again separated by commas, with all input data to be stored as `doubles`. (And, because we are working with `doubles`, values that differ from each other by less than 10^{-6} should be regarded as being equal.)

You may assume that each data line has exactly the number of fields specified by the words in the header row. For example, the following has six labeled columns and satisfies the input requirements:

```
year,month,day,location,mintemp,maxtemp
2024,4,28,18,6.7,12.9
2024,4,28,22,12.7,19.1
2024,4,29,18,7.6,15.3
2024,4,29,22,13.4,21.9
2024,4,30,18,7.3,21.8
2024,4,30,22,13.2,23.2
2024,5,1,18,9.4,15.9
2024,5,1,22,16.1,27.2
2024,5,2,18,8.7,16.3
2024,5,2,22,14.2,21.4
```

In this example, the first three fields describe a date in yyyy-mm-dd format, the fourth is a location flag ("`18`" for Melbourne, "`22`" for Sydney, perhaps), and the last two fields are measured temperature data. A five-day period at two locations is described.

Copy the skeleton file that is linked from the Assignment 2 LMS page. You will need to read through it carefully, and after you understand its structure will you be ready to add further functionality. There are several quite complex things provided in the skeleton (that you can choose to note, or choose to ignore):

- The use of `fopen()` and `fscanf()` in the function `read_csv_file()` to read data from the CSV file named as the first argument on the command line (use of the `fopen()` and `fscanf()` and etc functions will *not* be part of the assessment for this subject). Function `read_csv_file()` is finished, and you do not need to change it.
- The process of reading the CSV data into the structure `D`, counting the rows and columns, and also capturing the column headings in a string and breaking that string up in to parts to make an array of strings. Note that `read_csv_file()` converts any non-numeric fields in the CSV file into `nan` values so that any downstream computations will get infected. *You may assume that there will not be any non-numeric data in any of the data files used for assessment testing.*
- The way that `D` stores the data as an array of columns, with each column retaining its heading label via a pointer into a "snipped up" string that was originally the first input line.
- The function `reassign_input()`, which diverts `stdin` if there is a second file named on the command line, and sets `fileinput` to be true, which then causes the input to be echoed to `stdout`. The function `freopen()` will also *not* be examined in this subject.
- The way in which the main program loops, reading "commands" from `stdin` and "executing" them.

As well as reading in the CSV file, the skeleton program implements two "commands": typing "`i`" to the prompt "`>`" generates an "i"ndex listing of the fields in the input file; and typing "`d`" generates a complete "d"ump of the CSV data.

```
mac: ass2-soln test0.csv
Data file: test0.csv
    6 columns and 10 rows
> i
      col  data
        1  year
        2  month
        3  day
        4  location
        5  mintemp
        6  maxtemp
> ^D
ta daa!
mac:
```

Further example output will be shown in a lecture, and can be found on the LMS. To be clear: while the skeleton code makes use of several of the file manipulation functions described in Chapter 11 you do NOT need to understand how they work, and you will NOT be examined on those functions. Nor do you need to alter any of the functions that have already been completed.

Stage 1 – Column Averages (marks up to 8/20)

The skeleton program includes stubs for several further commands. The first of these is the "`a`" command, which computes the "a"verage value in the column specified by its argument, with column numbering always starting from one, as shown in the index listing. Implement the body of the function `do_averge()` without altering the function's interface to the rest of the program.

For the example data, you are aiming for:

```
> a 5
average mintemp:   10.93 (over 10 values)
    max mintemp:   16.10
    min mintemp:    6.70
```

All data values that are printed, and all values derived from data values, are to be printed to two decimal places using `%7.2f`.

Stage 1 requires that around a dozen lines (plus maybe a short function or two) get added to the program, and should be straightforward once you have determined where to add in the necessary code. The marks assigned to this stage are primarily a reward for you spending the time to read through the skeleton, and understand how it works. To obtain full marks you need to *exactly* reproduce the required output lines, and must also comply with the expectations in regard to style, layout, and general "program poetry". Further example input files and the required outputs are linked from the Assignment 2 LMS page.

A Grok "Assignment 2" project will be available shortly, and will have the same functionality as was provided for Assignment 1. You can also download the skeleton program and the test/output files and develop your program outside Grok if you wish.

Stage 2 – Graphing Distributions (marks up to 16/20)

Next, t "g" command generates a "g"raph of the values in a specified column. The two bounding values of that data column, `max` and `min`, should be computed, and then the range between `min` $- \epsilon$ and `max` $+ \epsilon$ broken in to 20 equal regions, where $\epsilon = 10^{-6}$. The number of values in each of the buckets is then computed, and plotted as a graph of at most 50 further columns of `'*'` characters. For the same test data, *but with the number of graph rows set to* 5 *rather than* 20 *to generate a shorter output for this handout*:

```
> g 6
graph of maxtemp, scaled by a factor of 1
  24.34--  27.20 [    1]:*
  21.48--  24.34 [    3]:***
  18.62--  21.48 [    2]:**
  15.76--  18.62 [    2]:**
  12.90--  15.76 [    2]:**
```

The horizontal scaling factor is the smallest power of two that gives rise to not more than 50 `'*'` characters in the longest output row in the graph. Rounding is upward: if the scaling factor is 8, then frequencies of 1 to 8 lead to one star, frequencies of 9 to 18 generate two stars, and so on. Use `%7.2f` for the two range values on each graph line, and `%5d` for the bucket count. Several output examples (with the correct number of rows and stars) are available on the LMS.

Stage 3 – More Functionality (marks up to 20/20)

First task: Command "c" computes "c"ategory averages, where the first column defines the categories:

```
> c 4 5
location   average mintemp
  18.00       7.94 (over 5 values)
  22.00      13.92 (over 5 values)
```

The output should be in ascending order of the category specified by the first column index. Any of the input columns can be used to define the categories, and any of the input columns can be the one that gets averaged. That means that you may *not* employ category values (such as 18 or 22) to index arrays, even if they are `ints` in the input file; you have to be more careful than that. You may assume that there will be at most `MAXCATS` categories required. Further output examples are available on the LMS.

Second task: The category averages can be helpful if the values in the column being used to create the divisions are discrete, for example, locations. But if the values are numeric, such as the temperatures, it is helpful to be able to compute a *correlation coefficient*. The “k” command computes Kendall’s tau-a correlation coefficient for n values, described as:

$$\tau_A = \frac{2(n_c - n_d)}{n(n-1)}$$

where

$$\begin{aligned} n_c &= \text{the number of concordant pairs of values} \\ n_d &= \text{the number of discordant pairs of values} \end{aligned}$$

A *concordant* pair is two rows of data where the values in the two specified columns are in same order; and a *discordant* pair is one where the two values are in reverse order. For example, in the dataset shown above, when comparing the final two columns, rows one and two (for April 28) are concordant, because $6.7 < 12.7$ and $12.9 < 19.1$. On the other hand, the April 29 and 30 rows for location 18 are discordant, because $7.6 > 7.3$, whereas $15.3 < 21.8$. In total, comparing the final two columns of the input data across all $10 \times 9 = 45$ combinations of two rows there are 36 concordant pairs, and 9 discordant pairs. Row pairs in which the values in either of the two columns are equal to each other (or differ from each other by less than 10^{-6}) are not counted as discordant nor counted as concordant. The Wikipedia[1] provides more information.

If two sets of paired values completely agree in their ordering, then a value of $\tau = 1$ will be reported. If they are opposite ordered, $\tau = -1$. When $\tau = 0$, there is no correlation between the two columns.

```
> k 5 6
tau coefficient between mintemp and maxtemp =    0.60
```

Refinements to the Specification

There may be areas where this specification needs clarification or even correction, and you should check the “Assignment 2” Ed Discussion page regularly for updates to these instructions. There is also a range of information linked from the “Assignment 2” LMS page that you need to be aware of.

The Boring Stuff...

This project is worth **20% of your final mark**, and is due at **6:00pm on Friday 17 May**.

Submissions that are made after the deadline will incur penalty marks at the rate of two marks per day or part day late, including for “weekend” days. Students seeking extensions for medical or other “outside my control” reasons should email `ammoffat@unimelb.edu.au` as soon as possible after those circumstances arise. If you attend a GP or other health care service as a result of illness, be sure to obtain a letter from them that describes your illness and their recommendation for treatment. Suitable documentation should be attached to **all** extension requests. Submissions will be completely closed exactly one week after the original deadline, and no extensions of more than seven days (including weekend days) will be granted.

Submission: Your `.c` file must be uploaded to GradeScope via the LMS “Assignment” page. *Don’t forget to include, sign, and date the Authorship Declaration that is required at the top of your program.*

Multiple submissions may be made; only the last submission that you make before the deadline will be marked. If you make any late submission at all, your on-time submissions will be ignored, and if you have not been granted an extension, the late penalty will be applied.

[1] `http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient`

Marking Rubric: A rubric explaining the marking expectations is linked from the assignment's LMS page, and you should study that rubric very closely. Feedback, marks, and a sample solution will be made available approximately two weeks after submissions close.

Academic Honesty: You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be individual work, not copied from anyone else, and not developed jointly with anyone else. So, do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your "Uni backup" memory stick to others for any reason at all; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" if they ask to see your program, pointing out that your "**no**", and their acceptance of that decision, are the only way to preserve your friendship. See `https://academicintegrity.unimelb.edu.au` for more information. Note also that solicitation of solutions via posts to "tutoring" sites or online forums, whether or not there is payment involved, and whether or not you actually employ any solutions that may result, is also serious misconduct. In the past students have had their enrolment terminated for such behavior.

The LMS page links to a program skeleton that includes an Authorship Declaration that you must "sign" and date and include at the top of your submitted program. Marks will be deducted (see the rubric linked from the LMS page) if you do not include the declaration, or do not sign it, or do not comply with its expectations. A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions. Students whose programs are identified as containing significant overlaps will have substantial mark deductions applied for failure to comply with instructions, or risk being referred to the Student Center for possible disciplinary action, without further warning.

Nor should you post your code to any public location (`github`, `codeshare.io`, etc) while the assignment is active or prior to the release of the assignment marks.

And remember, programming is fun!