School of Computing and Information Systems

comp20005 Intro. to Numerical Computation in C
Semester 1, 2023
Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of loops, `if` statements, functions, and arrays; writing a program that first reads some numeric data and then performs a range of processing tasks on it. The sample solution that will be provided to you after the assignment is completed will also make use of structures (covered in Chapter 8, and the Week 9 videos), and you may do likewise if you wish. But there is no requirement for you to make use of `struct` types, and the required tasks can equally well be carried out using parallel arrays.

Sequential Data

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, and then pass that set of parallel arrays (and a single buddy variable) into functions that carry out the required data transformations and analysis. Your task in this project is to use that processing approach to analyze some weather data that is provided as a time series. While this assignment uses one specific example of such data, the processing modes you will need to develop can be applied more generally as well. The data is from the City of Melbourne weather station at Argyle Square, near the University of Melbourne, and covers 2 June 2020 to 20 March 2023 (the date it was downloaded[1]), and then filtered to a smaller set of columns, and with data rows that had missing values also excluded.

For example, the first few lines `argyle-00050.tsv` contains these values in tab-separated format:

```
year    month   day     hour    minute  solar   wind    temp
2020    06      02      08      44      53      0.83    9.9
2020    06      02      09      05      77      1.29    10.4
2020    06      02      09      20      99      1.1     10.6
```

There will always be a single header line in all input files, and then rows each containing eight values separated by "tab" characters (`'\t'` in C). Once the first line has been bypassed (write a function that uses `getchar()` to read and discard characters until it has seen a newline character, `'\n'`), each data line can be read as a set of `int` and `double` variables using `scanf("%d%d%d%d%d%lf%lf%lf",...)`, with eight values per row:

- `year`, `month`, and `day`: as usual
- `hour` and `minutes`: the time within that day at which the measurement was taken
- `solar`: a measurement of solar energy at that time (in Watts per horizontal square meter)
- `wind`: the wind speed at that time (in meters/second)
- `temp`: the temperature at that time (in degrees Celsius)

All input files are sorted into date and time order. The measurements are taken at roughly fifteen minute intervals, but there are some times within each day that may have been missed for some reason (for example, power failures), and perhaps even whole days that got missed. In general, data files might contain dozens, hundreds, or thousands of lines. For this assignment you may assume that there will always be at least one line of data, and never more than 99,999 lines of data.

[1] See `https://data.melbourne.vic.gov.au/explore/dataset/meshed-sensor-type-1`

Stage 1 – Control of Reading and Printing (marks up to 8/20)

The first version of your program should read the entire input dataset into parallel arrays (or, if you are adventurous, an array of `struct`, but you are not required to do that), counting the data rows as they are read. The heading line should be discarded, and is not required for any of the subsequent processing steps. The required output of this stage for file `argyle-00050.tsv` is:

```
mac: ./ass1-soln < argyle-00050.tsv
S1, 50 data rows in total
S1, row     0: at 08:44 on 02/06/2020, solar =   53, wind =  0.83, temp =  9.9
S1, row    49: at 14:43 on 03/06/2020, solar =  198, wind =  2.00, temp = 13.0
S1, max solar =  433
S1, max wind  =  2.9
S1, max temp  = 14.5
```

Note that input *must* be read from `stdin` via "<" input redirection at the shell level. You *must not* make use of the file manipulation functions described in Chapter 11. No prompts are to be written.

To obtain full marks you need to *exactly* reproduce the required output lines. Two other examples can be found linked from the "Assignment 1" LMS page, and you can also create your own data files to test your program. You may assume that the input provided to your program will always be correct as described, and that you do not need to perform any data validation.

You may do your programming in `Grok`, and an "Assignment 1" project will be released shortly that includes the skeleton code and the test files. You should look at the handout "Running Programs in a Shell Within Grok", linked from the "Assignment 1" LMS page if you wish to do this. You should also read the instructions in the left-hand pane of the Grok "Assignment 1" project, to understand the automated testing and checking options (via `diff`) that are being used.

Or you may find it more convenient to use a separate programming environment on your computer, and download the skeleton program, test files, and expected output files to it. Information about both these options is available on the LMS.

Stage 2 – Building Tables (marks up to 16/20)

Ok, now for some computation. What we'd like to do is compute a table of average solar power outputs, where each row represents a period one hour long through the 24-hour day, and each column represents one month. The value printed in each cell of the table is the average (over how many observations appear in the data file for that `hour` and `month`) of the `solar` component in the input file. If there are no observations for some combination of hour and month, or if the average is less than 0.5, the string "`. .`" should be printed as the table entry.

The simplest way to compute the necessary table values is to use loops that iterate over `hour` (outer loop) and `month` (inner loop) and then for each of those combinations scan right through the data array looking for all of the matching entries. This is inefficient, but is still fast enough to be completely ok at the scale we are working, and you may follow this approach without being penalized. (Of course, your program should still execute moderately quickly, perhaps one second on the largest test file. If you have somehow made your program dreadfully inefficient, you will likely be penalized.)

There are three complete input/output examples linked from the LMS Assignment 1 page. If you are curious about the changing lengths of the days and the decrease and increase in solar power through the year, there is an interesting tool at `https://www.timeanddate.com/sun/australia/melbourne` that you might like to play with.

Stage 3 – Plotting The Trend (marks up to 20/20)

It can also be fun to visualize data patterns with simple character plots. The final part of your mission (should you decide to accept it!) is to capture the variation in temperature during the year.

Add further functionality to your program so that you generate a "sideways chart" that shows the range of observed temperatures in each of the twelve months. To do this, round each measured temperature to an integer, and compute the minimum and maximum values that occur in the data file for each given month (regardless of year). Once you have those month by month integer values `min_t` and `max_t` say, draw a "`****`" graph that has the first "`*`" in the `min_t`'th output column, and the last "`*`" in the `max_t`'th output column. If there are months with no observations in the data file (and hence no maximum or minimum defined), their output bars should be empty. You should also draw a scale across the top and bottom of the graph. Here is the output graph for the full $73{,}423$ data rows in `argyle-73423.tsv`:

```
S3,                  Min/Max Temperature by Month
S3,    -5    0    5   10   15   20   25   30   35   40   45   50
S3,     +----+----+----+----+----+----+----+----+----+----+----+
S3, Jan |                 ***************************          |
S3, Feb |               *******************************        |
S3, Mar |             ****************************             |
S3, Apr |           ************************                   |
S3, May |      ************************                         |
S3, Jun |       ****************                               |
S3, Jul |     ******************                               |
S3, Aug |       ********************                           |
S3, Sep |       ************************                       |
S3, Oct |         ************************                     |
S3, Nov |          ***************************                |
S3, Dec |             ********************************         |
S3,     +----+----+----+----+----+----+----+----+----+----+----+
S3,    -5    0    5   10   15   20   25   30   35   40   45   50

ta daa!
```

The output graph will always use the same scale and axes, and you can make then constant strings in your program.

You may again compute the required values by exhaustive scanning through the arrays storing the input data – that's what the sample solution does, and it processes this largest file in less than 0.1 seconds on a Mac laptop.

And whatever you do, don't overlook the final output line, it is also required!

Refinements to the Specification

There are bound to be areas where this specification needs clarification or even correction, and you should refer to the "Assignment 1" Ed Discussion page regularly and check for possible updates to these instructions. There is also a range of information linked from the "Assignment 1" 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 28 April**.

Submissions that are made after the deadline will incur penalty marks at the rate of two marks per day or part day late. 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.

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.

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!