School of Computing and Information Systems

comp20005 Engineering Computation
Semester 1, 2018
Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of loops and `if` statements by writing a program that sequentially processes a file of text data. You are also expected to make use of functions (Chapters 5 and 6) and arrays (Chapter 7, covered in lectures in Weeks 6 and 7). The sample solution that will be provided to you will also make use of structures (Chapter 8), and you may do likewise if you wish. But there is no requirement for you to make use of `struct` types, and they will not have been covered in lectures before the due date.

Sequential Data

Scientific and engineering datasets are often stored in text files using *comma separated values* (`.csv` format) or *tab separated values* (`.tsv` format), usually with a one- or two-line header describing the contents of the columns.

The data file used in this project was generated by the City of Melbourne, and uses `.tsv` format to record pedestrians as measured by counting devices at a number of locations around the CBD[1]. That data file was then edited to make the test files provided on the LMS. In particular, some of the columns were removed, some of the columns were reformatted, the hourly counts were aggregated into daily counts, and three recording stations were extracted into separate files, with the rest of the recording stations discarded. Smaller subsets of those three data files were also then prepared, see the LMS and Assignment FAQ page at `http://people.eng.unimelb.edu.au/ammoffat/teaching/20005/ass1/` for links to a full set of data files. The first few and last two lines of the file `pedestrians-melbcent-full.tsv` are:

```
# Melbourne Central Daily Pedestrian Counts
# yyyy  mm      dd      day     daycount
2009    06      01      2       22663
2009    06      02      3       22960
2009    06      03      4       23618
...
2018    02      27      3       33722
2018    02      28      4       33164
```

There will always be two heading lines in all input files, and then rows of five values separated by "tab" characters (`'\t'` in C). Once the first two lines have been bypassed, each data line can be read using `scanf("%d%d%d%d%d",...)`. The number in the fifth column is the pedestrian count for the day indicated by the `dd/mm/yyyy` date, with the fourth field, `day`, indicating the day of the week, numbered from Sunday to Saturday as integers 1 to 7 respectively. You can open these data files using `jEdit` if you wish to view them, and they can also be manipulated using `Excel`.

Stage 1 – Control of Reading and Printing (marks up to 4/10)

Your first program should read the entire input dataset into a collection of five parallel arrays (or, if you are adventurous, an array of `struct`), counting the data rows as they are read. The two heading lines should be discarded and not processed in any way.

[1]See `http://go.unimelb.edu.au/sht6` for a link through to the source dataset, which was accessed from `https://data.melbourne.vic.gov.au` on 28 March 2018.

Once the entire dataset is in memory, your program should print the first and last of the input records, and the total number of data lines that were read. The output from your program for this stage for file `pedestrians-melbcent-full.tsv` must be:

```
mac: myass1 < pedestrians-melbcent-full.tsv
S1: total data lines = 3067 days
S1: first data line  = 01/06/2009, 22663 people counted
S1: last data line   = 28/02/2018, 33164 people counted
mac:
```

Note that the input is to be read from `stdin` in the usual manner, via "<" input redirection at the shell level; and that you should *not* try and make use of the file manipulation functions described in Chapter 11.

You may (and should) assume that at most 10,000 days will be covered by the input data. Note that some of the marking process will be automatic, and you should seek to *exactly* reproduce these three output lines. Further examples can be found on the FAQ page linked from the LMS.

Stage 2 – Computing Stuff (marks up to 6/10)

Now add further functionality to your program so that it computes the number of entries there would be in the given date range if every day had a corresponding line in the data file. For the same data file, these additional lines should appear in your program's output:

```
S2: range spanned    = 3195 days
S2: coverage ratio   = 96.0%
```

The range spanned is to total number of days (including the two endpoints) in the date range that you reported in Stage 1, and the coverage ratio is the percentage of dates in this range for which the data file contains pedestrian count records. In this example, there are 3195 days between 1 June 2009 and 28 February 2018 (inclusive), and 3067/3195 is 96.0%. You can check date range calculations using Excel.

Note that code should be shared between the stages through the use of functions wherever it is possible to do so. In particular, there shouldn't be long (or even short) stretches of repeated or similar code appearing in different places in your program. Functions should also be used to break up long runs of code, to make them more understandable, even if those functions are only called once. As a general rule of thumb, functions shouldn't be longer than a single screen in the editor. If they are, break that code up further into smaller functions.

You may assume that the data records are presented in strictly increasing date order, and that you are *not* required to sort them. Note also that there will be no duplicate dates. On the other hand, you must *not* assume that there will be any particular year range in the input data, and must *not* assume that the months and dates will be exhaustive (there may be missing days, missing months and maybe even missing years).

Stage 3 – Make Summary Report (marks up to 8/10)

Modify your program further so that it also generates a "by the month" listing showing the average observed daily pedestrian numbers for each month for which there is any data provided in the input listing. Months that have no daily data provided should not be included in the output. For this stage, the required output for the test file `pedestrians-melbcent-p100.tsv` (with a total of 98 data lines) is:

```
S3: 06/2009 28/30 days covered, average count =  24.8k
S3: 07/2009 31/31 days covered, average count =  26.7k
S3: 08/2009 31/31 days covered, average count =  26.7k
S3: 09/2009  8/30 days covered, average count =  24.4k
```

where "k" means "thousands". See the LMS for the full output associated with the test files.

Stage 4 – Find the Trend (marks up to 10/10)

The Melbourne City Council is concerned that pedestrian traffic is increasing, and that the footpaths are becoming congested. To help them understand the growth rate over the sample period, further modify your program so that the input data is split into `NGRPS` near-equal sized (in terms of data records in the input file) groups of consecutive data lines from the input, where you should assume that `NGRPS` is to be three at present. The group sizes should be computed so that they differ by at most one from each other, and so that each input data line is in exactly one of the groups. There are several examples of the required output linked from the FAQ page.

The average number of pedestrians for each day of the week for each of the groups should then be computed, and plotted as a simple bar chart, to highlight any variations over the days of the week and over the `NGRPS` groups. For the data file `pedestrians-melbcent-full.tsv` the additional output from your program should be:

```
    S4: group 0 data, 01/06/2009 to 20/03/2012, 1022 data records
    S4: group 1 data, 21/03/2012 to 08/01/2015, 1022 data records
    S4: group 2 data, 09/01/2015 to 28/02/2018, 1023 data records

    S4: Sun, g0 = 22.2k |**********************
    S4: Sun, g1 = 27.6k |****************************
    S4: Sun, g2 = 28.8k |*****************************

    S4: Mon, g0 = 24.4k |************************
    S4: Mon, g1 = 27.6k |****************************
    S4: Mon, g2 = 28.3k |****************************

    S4: Tue, g0 = 24.9k |*************************
    S4: Tue, g1 = 28.6k |*****************************
    S4: Tue, g2 = 29.3k |*****************************
```

and so on, covering the other four days too, see the LMS for full output, where each `'*'` represents (correctly rounded off) one thousand people. Note that `NGRPS` should be a `#define` in your program, and that you should test your program using a range of values to ensure that the right output is produced.

Modifications to the Specification

There are bound to be areas where this specification needs clarification or correction. Refer to the FAQ page at `http://people.eng.unimelb.edu.au/ammoffat/teaching/20005/ass1/` regularly for updates to these instructions. There is already a range of information provided there that you need to be aware of, with more to follow.

Real Software

If you want to explore a much more sophisticated analysis tool for this pedestrian data, go via the redirection link `http://go.unimelb.edu.au/eht6` to a visualization tool authored by Boreak Silk.

The Boring Stuff...

This project is worth 10% of your final mark. A rubric explaining the marking expectations will be provided on the FAQ page.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. Submission will *not* be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software system known as `submit`. You can (and should) use `submit` **both early and often** – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. *Failure to follow this simple advice is highly likely to result in tears.* Only the last submission that you make before the deadline will be marked.

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. 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**" when they ask for a copy of, or to see, your program, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs are so identified will be referred to the Student Center for possible disciplinary action without further warning. This message* ***is*** *the warning.* See `https://academicintegrity.unimelb.edu.au` for more information. Note also that solicitation of solutions via posts to online forums, whether or not there is payment involved, is also taken very seriously. In the past students have had their enrolment terminated for such behavior.

Deadline: Programs not submitted by **10:00am on Monday 30 30 April** will lose 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 professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops in to something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Marks and a sample solution will be available on the LMS by Monday 14 May.

And remember, programming is fun!