School of Computing and Information Systems

comp20005 Engineering Computation
Semester 1, 2020
Assignment 1

Learning Outcomes

In this project you will demonstrate your understanding of loops, `if` statements, functions, and arrays, by writing a program that first reads a file of text data, and then performs a range of processing tasks on the data. The sample solution that will be provided to you after the assignment has been completed will also make use of structures (covered in 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 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 those arrays (and a buddy variable) into functions that perform various the required data transformations and analysis.

Your task in this project is to use that processing approach to determine shipping arrangements for the transport of a mineral reso alia in which huge machines di r overseas for processing. ration of iron ore in it, because natura dirt. Each scoop is transferred into a railway *wagon* at th ndreds of wagons are taken by rail from the mine to the port, for shipping to overseas s ore concentration in each wagon, samples are taken of the load at the time th and are analyzed for purity. Those samples are processed while the trai concentration in each wagon is known by the time each train of wagons reaches the port a few days later. The wagons are also weighed as they depart the mine, so that the amount of ore they contain is known.

To describe a train of wagons, a tsv file contains one line for each wagon in the train. The examples that follow, supposes that the data file `wagons0.tsv` contains these values in tab-separated format:

```
tonnes   percent
100.4    50.3
94.2     48.2
89.7     61.3
105.2    42.9
108.8    65.2
95.2     55.1
101.6    47.2
104.5    51.2
108.6    59.4
```

There will always be a single header line in all input files, and then rows containing pairs of values separated by "tab" characters (`'\t'` in C). Once the first line has been bypassed (write a function that reads and discards characters until it has read and discarded a newline character, `'\n'`), each data line can be read as a pair of `double` variables using `scanf("%lf%lf",...)`. The two values in each row

[1] The iron ore will eventually come back to Australia as cars and tv sets and steel beams, but that's another story.

represent a weight in tonnes, and the corresponding percentage ore concentration. This example file and another longer one `wagons1.tsv` can be copied from `http://people.eng.unimelb.edu.au/ammoffat/teaching/20005/ass1/`. The train shown in `wagons0.tsv` contains nine wagons/loads, and has a total weight of 908.2 tonnes of ore.

When a train of wagons arrives at the port they are combined to make *consignments*, where each consignment contains an *integral* number of wagons of ore. As the consignment is being formed, the overall weight and concentration are monitored. For example, if the first four wagons in `wagons0.tsv` were combined into a consignment, it would weigh $100.4 + 94.2 + 89.7 + 105.2 = 389.5$ tonnes and have an overall concentration of:

$$\frac{(100.4 \times 50.3\%) + (94.2 \times 48.2\%) + (89.7 \times 61.3\%) + (105.2 \times 42.9\%)}{(100.4 + 94.2 + 89.7 + 105.2)} \approx 50.3\% \, .$$

If each consignment must be a minimum of 375 tonnes at a minimum concentration of 50.0%, this four-wagon consignment can be *accepted*, and transferred to one of the waiting ships. But if the requirement was for 375 tonnes at 52.5%, this consignment would be *rejected*, and sold (at a loss) in a secondary market. Finally, if the minimum contracted size of the consignments was actually 400 tonnes, a fifth wagon would be need to be added, and the concentration would need to be recalculated before it could be known if the consignment could be accepted or rejected.

The clear goal is to avoid forming consignments that greatly exceed the minimum weight, or that greatly exceed the required concentration (because the company will not get paid for the excess minerals they contain); but at the same time also avoid forming consignments that fall below the required concentration (because the consignment will be rejected).

For all of the task in this assignment, consignments must be 375 tonnes or more, and to be accepted must have an ore concentration of 52.5% or more[2] (Be sure to `#define` these quantities!) We will also assume throughput t ach train arrives at the port, it rolls s

unloaded to one of two possi to add that wagon's load to the current consignment, or "tilt wagon right" to market. Hence, your program will model the situation in which only on time, and with the wagons being unloaded in the order that they appear in t will have multiple unloading stations and multiple consignments bei e program required much more complex, and beyond our expertise in this subject.

Stage 1 – Control of Reading and Printing (marks up to 5/10)

The first version of your program should read the entire input dataset into parallel arrays (or, if you are adventurous, an array of `struct`), 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. Once the entire dataset has been read, your program should print the first and last of the wagons' tonnages and concentrations (counting the wagons starting at "1"), the total of the wagon weights, and the overall ore concentration across the train. The output for this stage for file `wagons0.tsv` must be:

```
mac: ./myass1 < wagons0.tsv
S1,      wagon   1, tonnes=  100.4, %= 50.3
S1,      wagon   9, tonnes=  108.6, %= 59.4
S1,    whole train, tonnes=  908.2, %= 53.4
```

Note that the input is to be read from `stdin` in the usual manner, via "<" input redirection at the shell level; and that you must *not* make use of the file manipulation functions described in Chapter 11. No prompts are to be written. You may (and should) assume that each train contains at most 999 wagons[3].

[2] In reality, consignment sizes will be in the thousands of tonnes, and will be matched against ship capacities.

[3] The actual trains used in Western Australia are around 250 wagons long, but hey, we should plan for growth – computer memory is cheap. Some iron ore trains are also driverless, making them the world's biggest "robots". Unsurprisingly, when something goes wrong with an iron ore train, it goes really *really* wrong, see `https://tinyurl.com/y8v35spn`.

Note that to obtain full marks you need to *exactly* reproduce the required output lines. Full examples can be found on the FAQ page linked from the LMS. You can assume that the input provided to your program will always be sensible and correct, and you do not need to perform any data validation.

You may do your programming in your `grok` playpen, in which case you will probably wish to also create some test files there too, and will need to click on the "Terminal" button to run your program. Or, given the scale of the required program, may find it more convenient to move to the `jEdit/gcc` environment. Information about this option is available on the LMS.

Stage 2 – Simple Sequential Processing (marks up to 8/10)

In the simplest mode of operation, each train's wagons are combined in strict arrival order to reach the minimum weight, and then the concentration of that consignment calculated. If the concentration exceeds the required minimum, it can be loaded. On the other hand if the average concentration over the wagons in the consignment is less the the target purity, the consignment is rejected, and transferred to a dumping area. In both cases the next consignment is then commenced.

Add further functions to your Stage 1 program to reflect this mode of operation. Calculate the wagons that are joined to make each consignment when this (simple) strategy is employed, the tonnage and concentration of each consignment, and an overall summary of the ore tonnage and its concentration that do not get shipped, because of wasted wagons, or because of rejected consignments. The required output for this stage on `wagons0.tsv` is:

```
S2, consignment  1, tonnes=  410.1, %= 54.8
S2,                 wagons=    5   6   7   8
S2,   total dumped, tonnes=  498.1, %= 52.3
S2,                 wagons=    1   2   3   4   9
```

There are more examples l [illegible]

Stage 3 – Better Sequential Processing (marks up to 10/10)

The problem with the simple approach described for Stage 2 is that whole con [illegible] wasted, including wagons that individually meet the concentration r [illegible] the following approach:

- first, a group of wagons that make up the minimum consignment weight is identified
- then, their overall concentration is computed
- if that value is too low, the wagon with the lowest concentration is dumped, and the next wagon on the train is considered for inclusion in to replace it (or more than one more wagon, to meet the weight requirement)
- if the addition(s) make the consignment acceptable, it is confirmed, and the whole process repeats from the beginning
- or, if the revised consignment is still unacceptable, the (current) lowest concentration wagon is again marked for dumping, and another replacement wagon(s) is selected from the train.

On `wagons0.tsv` this strategy leads to this output:

```
S3, consignment  1, tonnes=  393.1, %= 56.4
S3,                 wagons=    1   2   3   5
S3, consignment  2, tonnes=  409.9, %= 53.3
S3,                 wagons=    6   7   8   9
S3,   total dumped, tonnes=  105.2, %= 42.9
S3,                 wagons=    4
```

More detailed examples are linked from the FAQ page. Pay careful attention to the last output line from each run; it is also a required part of your output.

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.

The Boring Stuff...

This project is worth 10% of your final mark. A rubric explaining the marking expectations is linked from the FAQ page, and you should read it carefully.

You need to submit your program for assessment; detailed instructions on how to do that are linked from the FAQ page. Submission will *not* be done via the LMS; instead you need to make use of a system known as `submit`, available at `http://dimefox.eng.unimelb.edu.au` (important: you *must be running the VPN in order to access this URL*). 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 the test server (a Unix computer called `dimefox`), which has some different characteristics to the lab machines. *Failure to follow this simple advice is likely to result in tears.* Only the last submission that you make before the deadline will be marked.

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. 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 a d their acceptance of that decision, are the only way t ictintegrity.unimelb. edu.au for more inform s, whether or not there is payment involved, is also Academic Misconduct. In the past st enrolment terminated for such behavior.

The FAQ page contains a link to a program skeleton that includes a t you must "sign" and include at the top of your submitted program. Marks will be deducted (see the rubric linked from the FAQ 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 penalties applied, or be referred to the Student Center for possible disciplinary action, without further warning.

Deadline: Programs not submitted by **11:00pm on Sunday 10 May** 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 25 May.

And remember, programming is fun!