School of Computing and Information Systems

comp20005 Engineering Computation
Semester 1, 2019
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. A standard processing framework on 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 computations based on the arrays.

Your task in this project is to compute delivery sequences for Alistazon, a start-up company that is planning to use drones to deliver parcels to customers. Each batch of deliveries is described by a file that has one row of numbers for each package that has to be delivered. In the examples that follow we suppose that the data file `drones0.tsv` contains these values in tab-separated format:

```
x       y       kg
150.6   -185.2  2.5
113.9   500.9   4.15
-31.6   51.9    4.31
-58.2   190.3   5.7
-27.8   312.6   1.95
```

There will always be a single header line in all input files, and then rows of three values separated by "tab" characters (`'\t'` in C). Once the first line has been bypassed (write a function that reads characters and throws them away until it reads and throws away a newline character, `'\n'`), each data line can be read using `scanf("%lf%lf%lf",...)`. The three values in each row represent an (x, y) location on a east-west/north-south grid, in units of meters relative to an origin, and a package weight, in units of kilograms. This example file and another one `drones1.tsv` can be copied from `http://people.eng.unimelb.edu.au/ammoffat/teaching/20005/ass1/`.

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

The first version of your program should read the entire input dataset into a collection of three 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 total number of data lines that were read, the first and last of the input records, and the total of the package weights. The output from your program for this stage for file `drones0.tsv` must be:

```
mac: myass1 < drones0.tsv
S1, total data lines:  5
```

```
S1, first data line :  x= 150.6, y=-185.2, kg=2.50
S1, final data line :  x= -27.8, y= 312.6, kg=1.95
S1, total to deliver: 18.61 kg
```

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 at most 999 data lines will occur in any input file, not counting the header line. Note that to obtain full marks you need to *exactly* reproduce the required output lines. Full output examples can be found on the FAQ page linked from the LMS.

Stage 2 – Sequential Processing (marks up to 7/10)

The Alistazon drones are battery powered, and can carry up to 5.8 kilograms each in addition to the 3.8 kilogram weight of the drone itself. (Future models may be able to carry more, but this is the current limit.) When the drone is carrying a payload of w kilograms, a fully-charged battery pack allows the drone to safely fly a total of $6300/(3.8+w)$ meters. That is, an unladen drone can safely travel a total of $6300/3.8 = 1657.9$ meters, whereas a fully laden drone can only safely fly $6300/(3.8+5.8) = 656.3$ meters. The drones fly at a constant speed of 4.2 meters/second, regardless of what load they are carrying, and always fly in a straight line from one specified location to another.

As an example, suppose that a fully-charged drone starts at location $(0, 0)$ and is to carry out the first delivery in the list in `drones0.txt`. The distance to be flown can be computed as

$$\sqrt{(150.6-0.0)^2+(-185.2-0.0)^2)} = 238.7 \text{ meters.}$$

On the outward trip, before the delivery takes place made, the drone plus package has a weight of $3.8 + 2.5 = 6.3$ kg, and hence the drone range is given by $6300/6.3 = 1000.0$ meters. That is, the outward trip to the delivery point will consume $238.7/1000.0 = 23.9\%$ of the available battery power. On the return trip back to the origin, after the delivery has been made, the drone is lighter, and consumes another $238.7/1657.9 = 14.4\%$ of the battery capacity. The round trip to complete the first delivery will have a flight time of $2 \times 238.7/4.2 = 113.7$ seconds, and will consume a total of $23.9 + 14.4\% = 38.3\%$ of a full battery charge.

Add further functionality to your program to carry out these calculations for each package, and to tell the operators when the battery pack needs to be changed. You should assume that each delivery commences at and returns to the origin $(0, 0)$, and that the packages are delivered in the same order they are listed in the data file. For `drones0.tsv`, your output from this stage should be:

```
S2, package=  0, distance= 238.7m, battery out=23.9%, battery ret=14.4%
S2, change the battery
S2, package=  1, distance= 513.7m, battery out=64.8%, battery ret=31.0%
S2, change the battery
S2, package=  2, distance=  60.8m, battery out= 7.8%, battery ret= 3.7%
S2, package=  3, distance= 199.0m, battery out=30.0%, battery ret=12.0%
S2, change the battery
S2, package=  4, distance= 313.8m, battery out=28.6%, battery ret=18.9%
S2, total batteries required:   4
S2, total flight distance=2652.0 meters, total flight time= 631 seconds
```

Note how the large demand associated with package 1 forces a battery change straight after package 0, even though less than half the battery capacity was used. In contrast, packages 2 and 3 are able to be delivered on a single battery.

Before implementing this step, you should read the rest of the specification, to understand what else will be required as your program develops. 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. It is also completely ok to have functions that are just one or two lines long.

Your program should print an error message and exit if any of the packages would require more than 100% of a fully charged battery to be delivered, or if any package weighs more than 5.8 kilograms.

Stage 3 – Non-Sequential Processing (marks up to 9/10)

Ok, so now let's try and make better use of each battery. Add further functionality to your program so that after each delivery has been completed, all of the remaining undelivered packages are considered in turn, and if any of them can be delivered using the current battery, that delivery is carried out next.

That is, for each fully charged battery, you should consider the entire list of packages in order, starting at the beginning, and each package that *can* be delivered using the power still available in that battery *should* be delivered. The battery is changed to a fresh one only when there are no remaining packages that can be delivered using the old battery.

The required output for `drones0.tsv` is (hint, hint) similar in format to the Stage 2 output:

```
S3, package=  0, distance= 238.7m, battery out=23.9%, battery ret=14.4%
S3, package=  2, distance=  60.8m, battery out= 7.8%, battery ret= 3.7%
S3, package=  3, distance= 199.0m, battery out=30.0%, battery ret=12.0%
S3, change the battery
S3, package=  1, distance= 513.7m, battery out=64.8%, battery ret=31.0%
S3, change the battery
S3, package=  4, distance= 313.8m, battery out=28.6%, battery ret=18.9%
S3, total batteries required:   3
S3, total flight distance=2652.0 meters, total flight time= 631 seconds
```

Stage 4 – A Further Generalization (marks up to 10/10)

Stages 2 and 3 assumed that all deliveries were to start from the origin point at $(0, 0)$, maybe because that is where the Alistazon warehouse is located. But a smart employee (that had taken comp20005 while studying at University) has pointed out that maybe it could be better to put a batch of deliveries into a van, drive to some more useful location, and then do the deliveries from that point. They suggest that an appropriate starting point could be found by computing

$$(\hat{x}, \hat{y}) = \left(\frac{1}{n} \sum_{i=0}^{n-1} x_i, \frac{1}{n} \sum_{i=0}^{n-1} y_i \right)$$

as a kind of "centroid" location of the n delivery locations (x_i, y_i). The same package delivery approach as was used for Stage 3 would then be followed. The required output for this stage for `drones0.tsv` is:

```
S4, centroid location x=  29.4m, y= 174.1m
S4, package=  0, distance= 379.2m, battery out=37.9%, battery ret=22.9%
S4, package=  2, distance= 136.6m, battery out=17.6%, battery ret= 8.2%
S4, change the battery
S4, package=  1, distance= 337.6m, battery out=42.6%, battery ret=20.4%
S4, package=  3, distance=  89.1m, battery out=13.4%, battery ret= 5.4%
S4, change the battery
S4, package=  4, distance= 149.8m, battery out=13.7%, battery ret= 9.0%
S4, total batteries required:   3
S4, total flight distance=2184.5 meters, total flight time= 520 seconds
```

The FAQ page contains links to full output examples that show how the output from the stages is to appear overall. Note the final output line at the end of each execution it is also required.

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 log in to a Unix server and make use of a 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 the test server, 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.

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. Students whose programs are identified as containing significant overlaps will be evaluated for mark penalties of for referral 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.

Very Important: *The FAQ page contains a link to a program skeleton that you must start with, including an Authorship Declaration that you must "sign" and include at the top of your submitted program. Marks will be deducted if you do not include the declaration, or do not sign it, or do not comply with its expectations.*

Deadline: Programs not submitted by **10:00am on Monday 6 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 20 May.

And remember, programming is fun!