School of Computing and Information Systems

comp20005 Intro. to Numerical Computation in C
Semester 1, 2024
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. You may also make use of structures if you wish (Chapter 8). But there is no requirement for you to make use of `struct` types, and the required tasks can equally well be carried out using a small number of one-dimensional arrays.

Background and Task

Imagine that the University of Melbourne has won the rights to host the first International Lecturing Olympics in 2024, and that a program is needed to read the judges' (that is, students') scores for each class given by each academic contender, and from them determine the medal winners.

Each input line to that scoring program will look like this:

```
A 5.5 6.6 5.2 5.8 8.1 7.2 8.0
```

in which:

- The first input value on each line is a single character (in the example, `'A'`) that is a code that identifies the lecturer that gave this class, and is to be stored as an `int` after being read into a `char` variable using `scanf("%c",...);`
- All remaining values in each line are student scores (between 0.0 and 10.0) from the students that attended that class (in the example there are seven such scores), separated by single spaces, and are to be stored as `double` and read using `scanf("%lf",...);` and
- The last score in each line is immediately followed by a newline character.

The input file consists of multiple such lines, one per class given during the Olympic competition. All input lines will be formatted exactly as shown, but with varying numbers of scores; and you do not need to try and anticipate variations to the input format, or to carry out any data validation. Several sample input files are linked from the LMS Assignment 1 page, and you can also create your own.

To compute the score for each class your program needs to implement the following steps:

- Input lines with fewer than three scores are discarded, and it is as if that class never happened; else
- The two smallest scores in each line are identified and discarded;
- The remaining scores are summed; and then
- The per-class "adjusted average score" is computed by dividing that sum by the original number of scores that were provided.

The example input line shown above would have the 5.2 and the 5.5 discarded, since they are the two lowest scores, and then the calculation $(6.6 + 5.8 + 8.1 + 7.2 + 8.0)/7 = 5.100$ would be done. This adjusted average calculation discounts the scores for small classes relative to large classes, compensating for the natural lecturing advantage of small classes.

Each competing academic gives several lectures through the course of the competition, with each of their classes recorded as an input line using their one-character code. The per-class scores for each academic are then collated into an overall per-lecturer score using a very similar computation:

- Lecturers with fewer than three per-class scores are discarded, and it is as if those lecturers didn't enter the competition; else
- The two smallest per-class scores for each lecturer are identified and discarded;
- The remaining per-class scores are summed; and then
- The per-lecturer score is computed by dividing that sum by the number of per-class scores associated with that lecturer.

This scoring regime allows each lecturer to have two "off" days when they need to present important but boring material (such as the scope rules in C); and rewards lecturers who take more (and better) classes. For example, suppose that lecturer `'A'` gave a total of six lectures during the Olympics, and that there are thus six input lines that start with `'A'`. Those six lines (see file `test1.txt`) translate into a set of five per-class scores, because one of the six classes doesn't have enough student scores to be used:

input lines (file `test1.txt`)	score computation	per-class score
`A 5.5 6.6 5.2 5.8 8.1 7.2 8.0`	$(6.6 + 5.8 + 8.1 + 7.2 + 8.0)/7$	5.100
`A 9.9 9.8 9.2`	$9.9/3$	3.300
`A 3.7`	*discarded*	–
`A 4.4 8.7 6.2 7.8 7.7 9.1 5.1 8.9`	$(8.7 + 6.2 + 7.8 + 7.7 + 9.1 + 8.9)/8$	6.050
`A 3.8 3.8 5.5 5.6`	$(5.5 + 5.6)/4$	2.775
`A 2.2 6.1 8.3 5.4 8.1`	$(6.1 + 8.3 + 8.1)/5$	4.500

Then, from the five per-class scores, the two smallest values get dropped and lecturer `'A'` gets a per-lecturer score of $(5.100 + 6.050 + 4.500)/5 = 3.130$ (to three decimals). Note the similarity of the per-class and per-lecturer computations, and take the hint – maybe there is a function to be developed for that task.

The per-lecturer scores are then basis for the final ranking. The lecturer with the highest score is awarded the gold medal, and the next two highest per-lecturer scores receive the silver and bronze medals. (But note that score ties must be checked, see Stage 3.)

You should assume that (and use `#define` statements for) the following limits on the input:

- There will be at most 26 competitors, with individual lecturer codes given by the single characters from `'A'` (which could be "Alistair", perhaps) through to `'Z'` (could be "Zoe").
- There will be at most 50 scores in any given input line (students who watch the lectures from home are not eligible to cast votes);
- There will be at most 500 lectures delivered in total (across all competitors) as part of the Olympic competition.

Stage 1 – Reading the Data (marks up to 8/20)

The first task is reading all the input data, and processing it into an array of per-class scores, with a lecturer code associated with each such score. As a check that you have read accurately, you should also print the last set of input values that were read and (if it has one) the corresponding per-class score. For the six lines shown above (the input file `test1.txt`), the required output for this stage is:

```
mac: ass1-soln < test1.txt
Stage 1, 6 class lines in input
Stage 1, 5 per-class adjusted scores retained
Stage 1, last line of input was
    "A: 2.2 6.1 8.3 5.4 8.1"
Stage 1, last per-class score was 4.500
```

If the last input score set contains fewer than three values, the final Stage 1 output line is not printed.

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. Further example input files and the required outputs can be found linked from the Assignment 1 LMS page.

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 this option is also available on the LMS.

Stage 2 – Aggregate by Lecturer (marks up to 16/20)

Ok, next you need to compute the per-lecturer averages, using the array of per-class scores (and the parallel array of associated lecturer codes) that you constructed in Stage 1. There are several input/output examples linked from the LMS Assignment 1 page that show what is required, and you should study them carefully. Note the row ordering: each lecturer has their overall adjusted average calculated and presented in the order in which they first gave rise to retained per-class scores.

Stage 3 – Gold, Silver, Bronze! (marks up to 20/20)

The final task is to determine the medal winners. The easiest way to do this is to sort the array of per-lecturer adjusted averages (from Stage 2) into decreasing order, and print the first three. You may use insertion sort, but be sure to also swap around the lecturer code that is associated with each per-lecturer score. Two helpful functions have been provided in the skeleton code.

Then the first (normally) three entries can then be printed as being medal winners, taking them from the beginning of the array. But if there are fewer than three lecturers with per-lecturer scores, fewer than three lines will be printed. You also need to be careful of ties, and if there are any further lecturers who get the same score (to three decimal places) as the third placed lecturer, they must also be reported. A human overseer will then determine what mixture of medals will be awarded in such cases. For example, if (to three decimal places) four lecturers all get the same top score, there might be four gold medals.

There is a further function provided in the skeleton code to help with this task, and file `test4.txt` provides an input example and shows what must be reported.

Finally, whatever you do, don't overlook the final output line after the Stage 3 outputs, it is also required!

Refinements to the Specification

There may be areas where this specification needs clarification or even correction, and you should check the "Assignment 1" Ed Discussion page regularly for 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 26 April**.

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.

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!