The University of Melbourne
School of Computing and Information Systems
COMP20005 Engineering Computation
Semester 2, 2020
Assignment 2
Due: 4:00pm Wednesday 28th October 2020

1 Learning Outcomes

In this assignment, you will demonstrate your understanding of arrays and structures, and use them together with functions. You will further practise your skills in program design and debugging.

2 The Story...

Loudspeakers[1] convert an electrical audio signal into a corresponding sound. They are basic units of both indoor and outdoor sound systems such as Hi-Fi sound systems and sound reinforcement systems. Loudspeaker placement (i.e., deciding where to place loudspeakers) plays a critical role in audience experience.

The task in this assignment is to design and implement a program that models the sound patterns across a given region equipped with a number of loudspeakers, and calculates any zones that perceive an insufficient sound strength.

You do *not* need to be an audio engineer to carry out this assignment. However, you do need the following background information. The *sound pressure level* ("sound level" for short) is a measure for the perceived sound strength. It is usually measured in a unit called *decibel* (dB)[2]. When measuring the sound level of a sound source, the distance needs to be taken into consideration. A distance of one metre (1 m) from the source is a frequently used default distance. Typically, if measured at one-metre distance, a pin dropping generates a 10 dB sound; a Hi-Fi speaker generates a 90 dB sound; a sound reinforcement speaker generates a 100 dB sound; and a jet engine generates a 150 dB sound. According to the *inverse proportional law*, given a sound source and its sound level L_1 as measured at distance r_1, then, at distance r_2, its sound level L_2 will become[3]:

$$L_2 = L_1 + 20 \log_{10} \left(\frac{r_1}{r_2}\right) \text{ dB} \qquad (r_2 \neq 0) \tag{1}$$

For example, when measured at 100 metres, the sound level of a sound reinforcement speaker (which has a sound level of 100 dB at one-metre distance) becomes 60 dB. *Note that sound level values must not be negative. If Equation 1 yields a negative number, L_2 should simply be made 0.*

A music festival organiser would like a tool that allows them to model the sound level in a given region equipped with a number of loudspeakers. The tasks described below lead you to the desired program. Note that the input format for different stages differs slightly, but that all lines in the input file will start with an uppercase letter. You must read the uppercase letter of each line, and process the line accordingly. Some stages only need some of the lines. *You may assume that between 1 and 99 loudspeakers will be specified, and that between 1 and 99 observation points will be specified.* No input validity checking is needed.

[1] `https://en.wikipedia.org/wiki/Loudspeaker`

[2] Sound levels may be measured using a frequency weighting filter, e.g., the A-weighting scale, which results in a measurement denoted by dB_A or decibels on the A-weighting scale. We simply use dB to ease the discussion.

[3] `https://en.wikipedia.org/wiki/Sound_pressure`. This equation requires certain conditions to apply, which might not suit our problem settings exactly. Nevertheless, we use it to simplify the discussion.

3 Your Task

3.1 Stage 1 - Arrays and Structs (Marks up to 5/20)

Write a program that reads a sequence of x, y, and sound level values from the standard input into an array of structures. Each relevant input line will start with an `S` in the first character position. It represents the location coordinates and the sound level of a loudspeaker. For example,

```
S 41.0 161.0 103.0
S 121.0 91.0 68.0
S 151.0 171.0 91.0
S 131.0 131.0 36.0
```

represent the locations of four loudspeakers with x and y coordinates in the positive quadrant of the plane measured in metres from the origin $(0, 0)$, and their sound level values (measured at one-metre distance) in decibel. *All input lines starting with an* `S` *will appear together at the beginning of the data file.*

Once the location and sound level data has been read, your program should calculate the *aggregate sound level* at the origin (location $(0, 0)$), by adding up the contributions of all the given loudspeakers using the following equation [4]:

$$L_{sum} = 10 \log_{10} \left(10^{\frac{L_1}{10}} + 10^{\frac{L_2}{10}} + ... + 10^{\frac{L_n}{10}}\right) \texttt{ dB} \tag{2}$$

Here, L_{sum} represents the aggregate sound level; L_1, L_2, ..., L_n represent the sound level contributed by n loudspeakers. Note:

- $L_{sum} \geq 0$; and
- if $L_i = 0$, it should not be added into Equation 2.

For the four loudspeakers (that is, $n = 4$) in the sample input, your output for this stage should be:

```
Stage 1
==========
Number of loudspeakers: 04
Sound level at (000.0, 000.0): 58.74 dB
```

Note that Equation 1 requires $r_2 \neq 0$. To guarantee the validity of Equation 1, the locations of the loudspeakers will *not* be shared with any of the points used in the following stages.

3.2 Stage 2 - More Loops (Marks up to 10/20)

The second block of input data are lines starting with a letter `O`, where each line contains the x and y coordinates of an observation point. Your program should read each of these lines, and compute the aggregate sound level at each observation point, summing over the loudspeakers' sound levels that were read in Stage 1 based on Equations 1 and 2. For example, if the next two input lines are:

```
O 43.0 31.0
O 147.0 129.0
```

Your Stage 2 output, following on from your Stage 1 output, would be:

```
Stage 2
==========
Sound level at (043.0, 031.0): 60.87 dB
Sound level at (147.0, 129.0): 63.69 dB
```

Note that all input lines starting with an `O` *will appear together after the lines starting with an* `S`.

[4] This equation has not considered the directions that the loudspeakers are facing. We use it to simplify the discussion.

3.3 Stage 3 - Yet More Loops (Marks up to 15/20)

To estimate the overall sound level of a given region, the simplest approach is to apply a regular grid, and evaluate the aggregate sound level at each point in the grid.

For this stage, assume that the region is a square of 312×312 metres, with edges perfectly aligned north-south and east-west. Add further functions and loops to your program so that it evaluates the sound level at every 4-metre grid point strictly within the region (that is, at the points $x \in \{4, 8, 12, ..., 308\} \times y \in \{4, 8, 12, ..., 308\}$, and counts the percentage of those points at which the aggregate sound level is lower than or equal to the 55 dB threshold which is considered to be too low. With the same four sample input shown in Stage 1, the next section of the output should be:

```
Stage 3
==========
5929 points sampled
0285 points (04.81%) have sound level <= 55 dB
```

3.4 Stage 4 - Draw a Map (Marks up to 20/20)

In this stage, you need to draw a "sound map" with a grid of characters. Each character displayed represents a cell of 12 metres wide (in the east-west direction) and 24 metres tall (in the north-south direction), with the 1:2 ratio (roughly) corresponding to the relative width and height of characters in a terminal font. To represent a square region of 312×312 metres, your map will be 26 characters wide and 13 characters high. To show the sound level contours, you use the following relationship between sound level in dB and the character displayed (where ' ' represents a whitespace character):

```
>=100   <100   <90   <80   <70   <60   <=55
'+'     ' '    '8'   ' '   '6'   ' '   '-'
```

The character plotted in each cell should correspond to the aggregate sound level at the centre of that cell. For example, the first value to be output (in the top left corner) should correspond to the aggregate sound level at the point $(x, y) = (6.0, 300.0)$, since each cell is taken to be 12 metres wide and 24 metres high. The bottom left corner of your map corresponds to a cell whose midpoint is at $(x, y) = (6.0, 12.0)$. Similarly, the top right and the bottom right points to be plotted are (306.0, 300.0) and (306.0, 12.0), respectively. A sample of the expected output for this stage is shown below. Note that there is a final newline character ('\n') at the end of the output.

```
Stage 4
==========
6666666                ---
66666666666             --
666666666666666          -
6666666666666666         -
66   666666666666
       66666 66666
  888  66666666666
6     66666666666
6666666666666666
666666666666666          -
6666666666666           --
6666666666             ---
                      ----
```

4 Submission and Assessment

This assignment is worth 20% of the final mark. A detailed marking scheme will be provided on Canvas.

See submission instructions on the next page.

You need to submit your code in Grok Learning (`https://groklearning.com`) for assessment. Submission will **not** be done via Canvas. Instead, you will need to:

1. Log in to Grok Learning using your student login details.
2. Navigate to the Assignment 2 module of our subject COMP20005 2020 S2: Engineering Computation.
3. Place your code in the `program.c` tab window.
4. Compile your code by clicking on the `Compile` button.
5. Once the compilation is successful, click on the `Mark` button to submit your code. (You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.*)
6. Two sample tests will be run automatically after you make a submission. Make sure that your submission passes these sample tests.
7. Two hidden tests will be run for marking purpose. Results of these tests will not be available until after the submissions are marked.

You can (and should) submit both **early and often** – to check that your program compiles correctly on our test system, which may have some different characteristics to your own machines.

You will be given a sample test file `test0.txt` and the sample output `test0-output.txt`. You can test your code on your own machine with the following command and compare the output with `test0-output.txt`:

```
mac: ./program < test0.txt    /* Here '<' feeds the data from test0.txt into program */
```

Note that we are using the following command to compile your code on the submission system (we name the source code file `program.c`).

```
gcc -Wall -std=c99 -o program program.c -lm
```

The flag "`-std=c99`" enables the compiler to use a modern standard of the `C` language – `C99`. To ensure that your submission works properly on the submission system, you should use this command to compile your code on your local machine as well.

You may discuss your work with others, but what gets typed into your program must be individual work, **not** from anyone else. Do **not** give (hard or soft) copy of your work to anyone else; do **not** "lend" your memory stick to others; 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.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **4:00pm Wednesday 28th October 2020** will lose penalty marks at the rate of three marks per day or part day late. Late submissions after 4:00pm Friday 30th October 2020 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer at jianzhong.qi@unimelb.edu.au. 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 into 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.

Special consideration due to COVID-19: Please refer to the "Special Consideration" section in this page: `https://students.unimelb.edu.au/student-support/coronavirus/exams-special-consideration-and-WAM`

And remember, *C programming is fun!*