The University of Melbourne
School of Computing and Information Systems
COMP20005 Engineering Computation
Semester 2, 2017
Assignment 2
Due: 4:00pm Wednesday 18th October 2017

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 the audience experience of sound systems.

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*, when a sound level L_1 is measured at distance r_1, then at distance r_2, the sound level L_2 is[3]:

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

For example, when measured at 100 metres, the sound level of a sound reinforcement speaker becomes 60 dB. *Note that a sound level value must not be a negative number. If Equation 1 yields a negative number, then 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; that between 1 and 99 observation points will be specified; and that the region boundary (Stage 5) consists of between 3 and 99 vertices.*

[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 51.0 71.0 90.0
S 101.0 51.0 90.0
S 35.0 71.0 100.0
S 51.0 201.0 95.0
```

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 that (1)* $L_{sum} \geq 0$*, and (2) if* $L_i = 0$ *then 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): 62.74 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 P, 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:

```
P 72.0 190.0
P 280.0 110.0
```

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

```
Stage 2
==========
Sound level at (072.0, 190.0): 68.05 dB
Sound level at (280.0, 110.0): 54.17 dB
```

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

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:

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

```
Stage 3
==========
5929 points sampled
1470 points (24.79%) 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 4 metres wide (in the east-west direction) and 8 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 metres, your map will be 78 characters wide and 39 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 $(x, y) = (2.0, 308.0)$, since each cell is taken to be 4 metres wide and 8 metres high. The bottom left corner of your map corresponds to a cell whose midpoint is at $(x, y) = (2.0, 4.0)$. Similarly, the top right and the bottom right points to be plotted are (310.0, 308.0) and (310.0, 4.0), respectively. A sample of the expected output is linked from LMS.

3.5 Stage 5 - Polygonal Boundaries (marks up to 20/20)

This stage is for a challenge. You do *not* need to complete this stage to obtain the full mark of the assignment. Before starting this stage, we suggest to submit (and preserve in a different directory) a program covering Stages 1 to 4 first.

There is 1 bonus mark for this stage. The bonus mark earned in this stage will compensate for the marks you lost in the earlier stages (assuming that you lost some, your total mark will not exceed 20).

Read a third segment of data, starting with a `V` at each line, which is followed by the x- and y-coordinates of a vertex on the boundary of the bounding polygon of a given region (Note: the last vertex should be connected back to the first vertex). For example, the input

```
V 0.0 0.0
V 80.0 300.0
V 200.0 175.0
V 275.0 55.0
```

describes a boundary that is an irregular quadrilateral. Your task is to print a second map that only shows the cells for which the cell centroid lies within the polygonal boundary; all other cells should be plotted as ‘`#`’. *Note that all input lines starting with a* `V` *will appear together after the lines starting with a* `P`.

Hint: you may assume that the polygon is convex, and hence that the centroid of the boundary points lies inside the polygon. A cell should be plotted only if the line segment between its centroid and the centroid of the polygon vertices does not intersect any of the boundary line segments. Hence, a critical component of your program needs to be a function that takes two line segments, described by two pairs of points, and determines if they touch in any way. That function will be supplied on the LMS page for the project, and you may copy and paste it into your program (and make sure to acknowledge the use of external code). Samples of the expected output will be given on the LMS page. There will of necessity be some overlap in control structure between this function and your Stage 4 function; that duplication will be tolerated.

4 Submission and Assessment

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

You need to submit all your code in one file named `assmt2.c` for assessment. The submission process is similar to that of Assignment 1. You will need to log in to a Unix server (dimefox.eng.unimelb.edu.au or nutmeg.eng.unimelb.edu.au) and submit your files using the following command:

```
submit comp20005 a2 assmt2.c
```

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. You should verify your submission using the following commands:

```
verify comp20005 a2 > receipt-a2.txt
more receipt-a2.txt
```

Note that, since the test output of this assignment is quite long, *we have simplified the verification output to only print out the lines where your submission output differs from our sample output.* If the `Diff` output for your submission is empty, it means that your submission output is the same as our sample output.

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: ./assmt2 < test0.txt    /* Here '<' feeds the data from test0.txt into assmt2 */
```

Only the last submission made before the deadline will be marked. You may discuss your work with others, but what gets typed into your program must be individual work, **not** copied from anyone else. Do **not** give hard copy 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 18th October 2017** will lose penalty marks at the rate of three marks per day or part day late. Late submissions after 4:00pm Friday 20th October 2017 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.

And remember, *C programming is fun!*