School of Computing and Information Systems

comp20005 Engineering Computation
Semester 1, 2020
Assignment 2

Learning Outcomes

In this project you will demonstrate your understanding of structures and arrays of structures, and will develop a computational solution for a non-trivial problem. You are expected to make extensive use of functions; and to demonstrate that you have adopted a clear and elegant programming style. You will find it difficult to create a working solution unless you plan your program carefully in advance, and develop it incrementally.

The Mission

An important design criteria in an outdoor carpark is to get an acceptable level of lighting (so that customers feel secure at night), with a small number of lighting towers (so that the cost of building them is not excessive) and appropriately powered bulbs (so that the cost of supplying electricity to them does not become excessive). Your task in this project is to design and implement a program that models the lighting patterns across an open outdoor carpark, and calculates any zones that are receiving an inadequate amount of lighting. Fortunately, the carpark is a rectangle, which makes our life much easier.

You need the following background information so as to be able to carry out this project. The light output of a bulb is measured in a unit called *lumens* which, for each different type of bulb, is related to the bulb's *Wattage*, which is a measure of the energy the bulb consumes. Lumens (SI abbreviation: lm) have units of candela $\cdot$ steradians, where a *candela* is a unit of light generation (now with a modern technical definition, but originally defined as being "the light emitted by a single candle"), and *steradians* are a unitless measure of two-dimensional arc coverage (as radians are to the circumference of a circle, steradians are to the surface of a sphere). Since a sphere of radius r has a surface area of $4\pi r^2$, a one candela source that radiates light equally in all directions generates an output of $4\pi \approx 12.6$ lm. A typical low-power 11 Watt compact-fluoro household light bulb generates around 400 lumens (for example, a bedside light); a bright household light generates around 1250 lumens (for example, a roof light in a kitchen); and a high-powered street lamp perhaps 5–10 times that much.

Perceived light intensity, or *illuminance*, is measured in *lux* (SI abbreviation: lx), which has units of lm/meter^2. Provided that there is line-of-sight to a light source, the illuminance at a distance of r meters from a source of ℓ lumens is computed as $\ell/(4\pi r^2)$. For example, a desktop that is 2 meters below a 400 lumen bulb receives an illuminance of approximately 8.0 lx. This is plenty to "see" by, but at the lower limit of what is required to read the newspaper by, and hopeless for detailed work such as embroidery, surgery, or watch repairs. Workplace health requirements typically mandate around 400 lx at the desktop, to avoid the risk of eye-strain. A grey and cloudy day provides around 2000 lx, and bright noon sunlight is around $100{,}000$ lx.

Your employer, AliMoParking UnLimited, uses a standard lighting tower in all of their carparks, deploying bulbs of differing output (in lumens), depending on the situation. Each standard tower is 8.25 meters high, and supports a single semi-spherical bulb at that height that radiates light evenly in all downward directions, as shown in Figure 1. For example, the point 8.25 meters immediately below a bulb that is "half-sphere" rated at $10{,}000$ lm receives an illuminance of $10^4/(4 \cdot \pi \cdot 8.25^2) \approx 11.7$ lx; and a point 15 meters from the pole (see Figure 1) receives $10^4/(4 \cdot \pi \cdot (8.25^2 + 15^2)) \approx 2.7$ lx[1].

AliMoParking UnLimited have a design policy that says no location in their carparks should have an illuminance of less that 1.0 lx, which is around ten times brighter than the full moon. To help realize

[1] Note that `M_PI` is not available in the math header file on the server; you should define your own constant and use the value 3.14159.

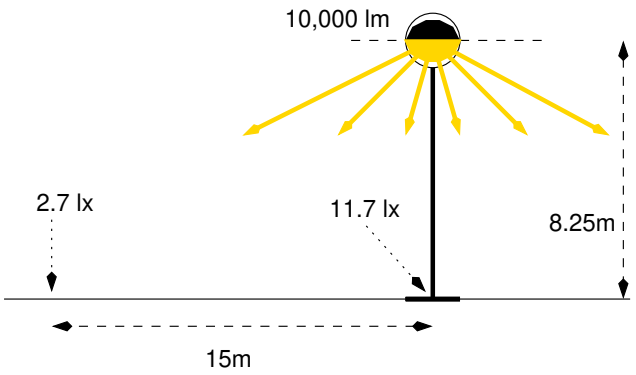

Figure 1: The AliMoParking standard lighting tower, when fitted with a 10,000 lm half-sphere bulb.

this goal, they would like a tool that allows them to model carpark lighting. The sequence of tasks leads you to the desired program. Students unsure of their programming skills should probably tackle the stages in the order they are presented. Confident students might wish to progress directly to Stage 3, but should still create their program incrementally. You may assume that at most 99 lighting towers will be specified. You should also assume that the light poles have zero width and do not cast shadows.

Stage 1 – Control of Reading and Printing (marks up to 10/20)

Each input file will consist of lines of floating point values, with *two* included header rows. For example, the file `lights0.tsv` contains:

```
maxx    maxy
45.0    90.0
x       y       lm
0.0     16.5    5000.0
0.0     67.0    8500.0
25.0    0.0     8500.0
50.0    35.0    8500.0
```

Each data file will commence with the (x, y) location in meters of the north-east (top-right) corner of a car park, with $x > 0$ and $y > 0$, and with the south-west (bottom-left) corner always assumed to be at $(0, 0)$. There will always be exactly one data line of this type (plus its one-line header) in each input file. There will then be a second header, and then a set of (x, y, ℓ) triples, with each triple describing one standard AliMoParking lighting tower, and with all distances measured in meters. In `lights0.tsv` there are four towers: two on the west boundary of the carpark (the y axis), one on the south boundary (the x axis); and one just outside the east boundary. Note that lighting towers might be inside the carpark, on the boundary of the carpark, or even outside the carpark (AliMoParking will always include nearby street lighting, to "borrow" light that is being paid for by someone else).

This stage of your program should read the boundary limits, then read all of the light tower descriptions into a suitable array of `struct`. It should then print out a summary, to confirm that the data has been read correctly. The required output is:

```
S1, north-east corner at x=  45.0m, y=  90.0m
S1, number of lights =  4
S1, total light output =  30500lm
```

Stage 2 – Processing Over A Grid (marks up to 15/20)

Now you need to work out if the proposed lighting arrangement is suitable. Implement loops that carry out an exhaustive *grid search*. To do the search, break each of the x and y dimensions into $100 \times$

100 equal-size small rectangular regions. There are to be 10,000 rectangular cells in total; and for `lights0.tsv` the south-west (bottom-left) cell will start at $(0, 0)$, and each cell will be 0.45×0.90 meters. Then, for the point location at the *centroid* of each cell, compute the illuminance (by summing up the illuminances derived from each of the lighting towers), and count the number of centroid points at which the illuminance falls below the bound of 1.0 lx. That is, the first point (for `lights0.tsv`) at which you compute the total illuminance will be $(0.225, 0.45)$. Express that total count of inadequate lighting as a percentage, so that for `lights0.tsv` you can compute this output:

```
S2, 100 x 100 grid, 0.45m x 0.90m cells
S2, fraction of cell centroids below 1.0lx requirement =  6.2%
```

(The fraction you have computed is, of course, an approximation. But if the grid was made 1000×1000 instead, it would be a pretty good one.)

Stage 3 – Having Some Fun (marks up to 18/20)

Then your boss[2] says, "well, thanks for that, but where are the dark spots, we need to know where they are so that we can illuminate them, can you draw a map?" To do that, you note that in a typical fixed-width terminal font such as `Courier` and with typical line spacing, each character is about 1.8 times taller than it is wide. To create a map that is 72 characters wide (to fit within a typical terminal window) for `lights0.tsv`, the following steps are performed:

- calculate the cell width as $45.0/72 = 0.625$ meters (where 45.0 is the carpark width);
- calculate the target cell height as 1.8 times that, ≈ 1.125 meters;
- divide the total map height of 90.0 meters by that, and round to the nearest integer, in this case the exact ratio is 80.000, so the integer is 80;
- use that integer as the number of vertical cells to appear in the map, and recalculate the cell height based on it, in this case, $90.0/80 = 1.125$ meters;
- use that value as the cell height, to create a map that is (always going to be) 72 characters wide and (for this particular carpark) 80 lines tall.

Each cell in the graph again has the lux value computed at the cell centroid. That computed value is then plotted as a single character via the mapping below. You may assume that the plot will never end up being more than 200 cells tall. Full examples are linked from the FAQ page.

< 1.0	< 2.0	< 3.0	< 4.0	< 5.0	< 6.0	< 7.0	< 8.0	< 9.0	< 10.0	≥ 10.0
'-'	' '	'2'	' '	'4'	' '	'6'	' '	'8'	' '	'+'

Stage 4 – The Last Two Marks (marks up to 20/20)

VERY IMPORTANT: You do not have to do Stage 4 to get a good mark for this assignment! Please tackle it only if you have already submitted a perfect Stage 3 program, and you want a serious challenge to try and get the last two marks. Note that I will be very selective in regard to giving any advice for this stage, and my most common answer will be "well, sounds to me like you still have some thinking to do".

Your boss likes the map, but then says (in a typically annoying request), "if we were to add one light tower into each of the low-light regions, how many more light towers would we need?" If a "low-light region" is defined as "low-light grid cells (in Stage 3) that are touching horizontally, vertically, or diagonally", count and report the number of low-light (less than 1.0 lx) regions that appear. For example, the plot

```
---    22222222222      --   -----         --         ---            22222222
----       22222222222   -----                -------------            22222222
--           222222       --                     --               22222222
```

has three distinct low-light regions. See the FAQ page for full examples.

[2] An annoying person called AliMo, who is always saying stupid things like "programming is fun" and "submit early and submit often", and who thinks that the long-term carpark at Melbourne Airport is one of the city's most beautiful places.

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/ass2/` 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 20% 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 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 online forums, whether or not there is payment involved, is also Academic Misconduct. In the past students have had their enrolment terminated for such behavior.

The FAQ page contains a link to a program skeleton that includes an Authorship Declaration that 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 31 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 assignment extension requests.

Marks and a sample solution will be available on the LMS by Monday 15 June.

And remember, programming is fun!