Department of Computing and Information Systems

COMP20005 Engineering Computation
Semester 1, 2013
Project 2

Learning Outcomes

In this project you will demonstrate your understanding of arrays and structures, and functions that make use of data stored in them. You will also build your skills in terms of program design, and testing and debugging.

The Story...

An important design criteria in any 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 and 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 calculate any zones that are receiving an inadequate amount of lighting.

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* (as distinct from the usual *Watts* that you see on the bulb's packaging, which is a measure of the energy the bulb consumes). Lumens (SI abbreviation: lm) have units of candela·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 solid-arc (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 $4\pi \approx 12.6$ lumens. A typical 11W compact-flouro household light bulb generates around 500 lumens; and a high-powered streetlamp perhaps 10–30 times that much.

Perceived light intensity, or *illuminance*, is measured in *lux* (SI abbreviation: lx), which has units of lumens/meter^2. Provided that there is a line-of-sight between a surface and the light source, the illuminance of a light source of ℓ lumens at a distance of r meters is computed as $\ell/(4\pi r^2)$. For example, a desktop that is 2 meters below a single household bulb of 500 lumens receives an illuminance of approximately 9.9 lux. This is plenty to "see" by, but marginally sufficient to do detailed work (such as watch repairs), or to read fine print. Workplace requirements typically require around 400 lux at the desktop; and bright sunlight is around $100{,}000$ lux.

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

CarpArking UnLimited have a design policy that says no location in their carparks should have an illuminance of less that 1.0 lux, which is something like the light of a full moon. To help them realize this goal, they would like a tool that allows them to model carpark lighting. The sequence of tasks described shortly 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 4, but should still create their program incrementally.

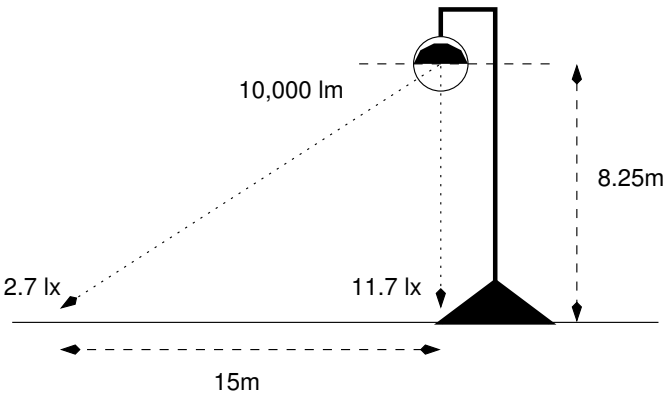

Figure 1: One lighting pole.

Note that the input format for each task differs slightly, but that all lines in the input file will be tagged. You must read the tag character at the start of each line, and process the line accordingly. Some of the stages only need some of the lines. You may assume that at most 100 lighting towers will be specified; that at most 100 other points will be specified; and that the carpark boundary (Stage 5) has at most 100 points on it. It is also assumed that the poles have zero width, and hence do not cast shadows.

Stage 1 – Arrays and Structs (marks up to 8/20)

Write a program that reads a sequence of x, y, and ℓ values from the standard input into a suitable declared array of structures. Each relevant input line will be tagged with a `P` (for pole) in the first character position. Lines that do not contain a `P` should be bypassed. For example,

```
P  5.0 75.0 8000.0
P 10.0 50.0 5000.0
P 35.0 75.0 3000.0
P 50.0 20.0 5000.0
```

represents the locations of four light bulbs as x and y coordinates in the positive quadrant of the plane and measured in metres from the origin $(0, 0)$, and their light output in lumens. You may assume that all input lines tagged with a `P` form a single block that appears at the beginning of the data file.

Once the pole location/intensity data has been read, your program should calculate the aggregate light intensity at the origin (location $(0, 0)$), by adding up the contributions of all of the specified lighting towers. For the four towers in the sample data, your output for this stage should be:

```
Stage 1
=========
Number of light poles: 4
Illuminance at ( 0.0, 0.0):  0.43 lux
```

Stage 2 – More Loops (marks up to 12/20)

The second block of input data is a set of x and y locations, in lines tagged with an `L`, two values per line. Your program should process each of these locations, and compute the illuminance at each designated point, summing over the poles that were read in Stage 1. For example, if the next two input lines are

```
L 73.0 19.0
L 28.0 11.0
```

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

```
Stage 2
=========
Illuminance at (73.0,19.0):  0.88 lux
Illuminance at (28.0,11.0):  1.03 lux
```

If there are no L-tagged data lines in the input file, no output should be generated for Stage 2.

Stage 3 – Yet More Loops (marks up to 15/20)

To estimate the extent to which the lighting planned for the carpark is inadequate, the simplest approach is to apply a regular grid, and evaluate the illuminance at each point in the grid.

For this stage, assume that the carpark is a square of 78×78 meters, with edges perfectly aligned north-south and east-west.

Add further functions and loops to your program so that it evaluates the illuminance at every whole-number-of-meters grid point strictly within the carpark (that is, at the points $x \in [1 \ldots 77] \times y \in [1 \ldots 77]$, and counts the percentage of those locations at which the lighting is below the 1.0 lux threshold that is deemed to be "inadequate".

With the same four sample input lines shown in Stage 1, the next section of the output should be:

```
Stage 3
=========
With 4 poles in use and 5929 points sampled, 1761 points (29.7%)
have illuminance below the 1.0 lux acceptable level
```

Stage 4 – Draw a Map (marks up to 19/20)

Having finished your program for Stage 3, you proudly show it to your boss, only to have them say "that's no use at all, you need to tell me where the dark spots are".

So you decide that the best way to show what is going on is to generate a plot of characters on the screen, as a kind of crude map. Each character displayed represents a cell one meter wide (in the east-west direction) and two meters 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. For the carpark in question, your plot is thus 78 characters wide and 39 characters high.

To show the lighting contours, you use the following relationship between illuminance in lux and the character displayed:

< 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'	' '	'+'

The character plotted in each cell should correspond to the illuminance at the center of that cell. For example, the first value to be output (in the top left corner) should correspond to the illuminance at the point $(x, y) = (0.5, 77.0)$, since each cell is taken to be one meter wide and two meters high. The bottom left corner of your map corresponds to a cell whose midpoint is at $(x, y) = (0.5, 1, 0)$. A sample of the expected output is linked from the FAQ page.

Stage 5 – Polygonal Boundaries (marks up to 20/20)

Considering doing this stage? It would be smart to submit (and preserve in a different directory) a "completely correct" program covering Stages 1 to 4 first, before even *thinking* about this last mark.

Read a third segment of data, tagged with `B` in the first column, where each input line contains an x and a y describing a vertex on the boundary of the bounding polygon. The last segment connects the final input point back to the first one. For example, the input

```
B  0.0  0.0
B 10.0 75.0
B 60.0 75.0
B 75.0 15.0
```

describes a boundary that is an irregular quadrilateral.

If (and only if) a boundary is supplied, you should 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 `'#'`. 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 polygon centroid 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 FAQ page for the project, and you may include it in your program. Samples of the expected output appear 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. (Though there is a way of abstracting away this repetition too. If you are really keen, see function arguments in Chapter 10!)

The boring stuff...

This project is worth 20% of your final mark.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. Submission will *not* be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software 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 our test system, which has some different characteristics to the lab machines. 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. In particular, note that we have access to sophisticated similarity checking software that undertakes deep structural analysis of C programs, and routinely run an automated check over all student submissions. So, 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.

Deadline: Programs not submitted by **9:00am on Monday 27 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 Alistair, `ammoffat@unimelb.edu.au`.

And remember, programming is fun!