Facility Location Problems

Phone companies want to find the right places to locate their cell-network towers; hamburger chains want to find the right locations to build new drive throughs; governments want to put schools in the right locations; and city councils want to locate recycling facilities at points where they will be used. In general, we want facilities to be conveniently located, so that we can access them when we need them. In this assignment we will be looking at the locations of shopping centers, but the principles involved apply to many services and facilities.

Program input will again be a *tab separated values* file. For example, `malls1.tsv` contains:

```
name    xloc    yloc    glam    carp
f       6.3     15.2    13.6    1.9
u       17.5    3.4     13.3    0.8
n       -8.3    -6.3    8.9     0.2
i       -11.5   -1.5    16.3    0.3
y       0.5     -0.5    9.3     1.1
```

where `name` is a letter that identifies the shopping center; `xloc` and `yloc` are the (x, y) locations in kilometers of a shopping center relative to the center of the city; `glam` is its *glamour score*, a measurement of the attractiveness of that shopping center, see below; and `carp` is a measure of the difficulty of getting a carpark at that shopping center. The city has as many as fifty different shopping centers.

Stage 1 – Control of Reading and Printing (marks up to 8/20)

You can base your program on your Assignment 1 solution, to get started. But note, for this assignment you are required to define and use a `struct` to store the input data. In this first stage, you have to show that you have successfully read the input data; on the file `malls1.tsv` the required output is

```
S1, 5 malls described in input file
S1, mall 0 = f, xloc =   6.3, yloc =  15.2, glam = 13.6, carp =  1.9
S1, mall 1 = u, xloc =  17.5, yloc =   3.4, glam = 13.3, carp =  0.8
S1, mall 2 = n, xloc =  -8.3, yloc =  -6.3, glam =  8.9, carp =  0.2
S1, mall 3 = i, xloc = -11.5, yloc =  -1.5, glam = 16.3, carp =  0.3
S1, mall 4 = y, xloc =   0.5, yloc =  -0.5, glam =  9.3, carp =  1.1
```

To obtain full marks you need to *exactly* reproduce the required output lines. Full examples can be found on the FAQ page linked from the LMS. The input provided to your program will always be sensible and correct, and you do not need to perform any data validation.

Stage 2 – Walking a Grid (marks up to 16/20)

The *relative attractiveness* of a shopping center to a resident of the city is determined by

$$RA = \frac{glamour}{distance + carpark},$$

where *distance* is how far that person lives from the shopping center, calculated in kilometers based on Euclidean straight-line travel (we might also consider using rectilinear distances if the road network contains roads that only run north-south and east-west, but that is for another day).

Suppose that each resident always goes to the shopping center that has the greatest *RA* value, based on where they live. So shopping centers that have higher glamour scores tend to draw people from suburbs that might actually be closer to a less glamorous shopping center. People are also prepared to drive further if the car parking is easier. For example, in `malls1.tsv` shopping center `f` is reasonably glamorous, but its carparking situation is not so good.

No resident will go to any shopping center if its *RA* is below 1.0 at their address. This may mean that some residents can't go shopping, and have to rely on deliveries. Your program is intended to help identify the business opportunities that these "no shopping nearby" spots create.

Suppose that the city stretches across a square range defined by $x \in [-25.0, +25.0]$ kilometers and $y \in [-25.0, +25.0]$ kilometers, and that we wish to compute the "attraction zone" of each shopping center as a percentage of the total space occupied by the city (which is 2500 square kilometers).

To do this, we split the city into 0.1×0.1 kilometer grid cells (that is, a grid containing $500 \times 500 = 250{,}000$ cells), and compute the preferred shopping center at the midpoint of each such cell. For example, the south-west cell spans the area $[-25.0, -24.9] \times [-25.0, -24.9]$, and is represented by its central point at $(-24.95, -24.95)$. By computing a *RA* value for each known shopping center at that point, we can determine which shopping center that point is attracted to, and then count that whole grid cell as going to the same center. This will then give an approximation of the desired percentage attraction; if we wanted to, we could decrease the edge size of each grid cell to get a more precise estimate.

The required output from your program from this stage for `malls1.tsv` is:

```
S2, 250000 grid cells processed, each 0.010 km^2
S2, mall f attracts  343.6 km^2 =  13.7% of city
S2, mall u attracts  360.3 km^2 =  14.4% of city
S2, mall n attracts   61.8 km^2 =   2.5% of city
S2, mall i attracts  614.3 km^2 =  24.6% of city
S2, mall y attracts  124.4 km^2 =   5.0% of city
S2, no shopping for  995.6 km^2 =  39.8% of city
```

Stage 3 – Drawing a Grid (marks up to 19/20)

Your boss is very pleased to know that around 40% of the city area does not have a good shopping center handy, and plans to invest in a new one and capture more of the shopping market. So for your next task, you are asked to generate a map showing the empty locations. So you switch to a different grid, now one in which each cell is 1 km wide and 2 km tall, and compute the coverage regions again. But this time, instead of adding them up, you draw them as a "character map", see Figure 1.

Once you have printed this you understand why each cell is twice as tall as it is wide – it approximates the aspect ratio of typical `terminal`-style fixed-width fonts. (Best of all, when you boss sees your output they have a nostalgia trip back to their own programming classes in the 1980s when they were nineteen and learnt C from Alistair, and award you an immediate pay rise based on anticipated future profits.)

```
S3,         +--------------------------------------------------+
S3,  24.0km |                        fffffffffffff             |
S3,  22.0km |                      fffffffffffffffff           |
S3,  20.0km |                     fffffffffffffffffff          |
S3,  18.0km |                    ffffffffffffffffffff          |
S3,  16.0km |                    ffffffffffffffffffff          |
S3,  14.0km |          iiiiiii   fffffffffffffffffuuuuuuuu     |
S3,  12.0km |     iiiiiiiiiiiiiiiffffffffffffffffuuuuuuuuuu    |
S3,  10.0km |  iiiiiiiiiiiiiiiiiiffffffffffffffuuuuuuuuuuuu    |
S3,   8.0km | iiiiiiiiiiiiiiiiiiiiffffffffffuuuuuuuuuuuuuu    |
S3,   6.0km |iiiiiiiiiiiiiiiiiiiiiyyyyfffffuuuuuuuuuuuuuuu    |
S3,   4.0km |iiiiiiiiiiiiiiiiiiiiiyyyyyyyyyuuuuuuuuuuuuuuu    |
S3,   2.0km |iiiiiiiiiiiiiiiiiiiiiyyyyyyyyyuuuuuuuuuuuuuuu    |
S3,   0.0km |iiiiiiiiiiiiiiiiiiiiiyyyyyyyyyuuuuuuuuuuuuuuu    |
S3,  -2.0km |iiiiiiiiiiiiiiiiiiiiiyyyyyyyyyuuuuuuuuuuuuuuu    |
S3,  -4.0km |iiiiiiiiiiiiiiiinniiiiyyyyyyyyyuuuuuuuuuuuuuuu    |
S3,  -6.0km |iiiiiiiiiiiiinnnnnnnniiyyyyyyyy  uuuuuuuuuuuuuu    |
S3,  -8.0km |iiiiiiiiiiiiinnnnnnnniiiiiyy        uuuuuuuuuu  |
S3, -10.0km |iiiiiiiiiiiiinnnnnnnniiiii                      |
S3, -12.0km | iiiiiiiiiiiiiinnnnniiiii                       |
S3, -14.0km |    iiiiiiiiiiiiiiiii                           |
S3, -16.0km |       iiiiiiiiiiii                             |
S3, -18.0km |                                                |
S3, -20.0km |                                                |
S3, -22.0km |                                                |
S3, -24.0km |                                                |
S3,         +--------------------------------------------------+
```

Figure 1: The required Stage 3 output.