The University of Melbourne
School of Computing and Information Systems
COMP20005 Engineering Computation
Semester 2, 2022
Assignment 2
Due: 4:00 pm Friday 21st October 2022
Version 1.0

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...

Electric vehicles (EVs), such as Tesla's S3XY models, are gaining popularity and attracting substantial public interest, especially in the recent fuel price hike, for their benefits in reducing tailpipe emissions, lower air and noise pollution, and more affordable running and maintenance costs.[1] Tens of thousands of EVs now run on Australian road networks. With the rising number of EVs comes the need for more charging stations.

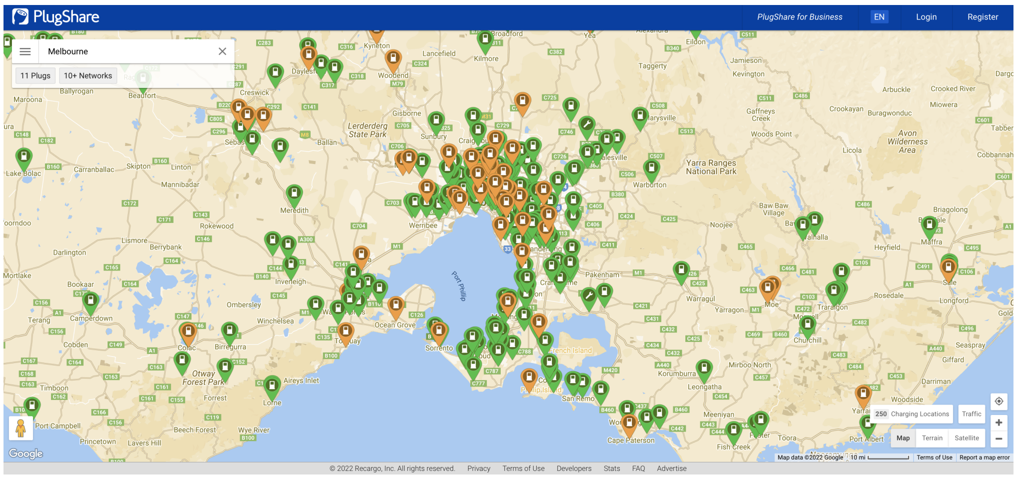

Figure 1: Electric Vehicle Charging Stations across Victoria (www.plugshare.com)

Figure 1 shows the distribution of EV charging stations across Victoria. While there may seem to be more charging stations than you might have expected, these are far from sufficient to accommodate for the EV charging needs. This is because of the long charging time needed for an EV – even the Tesla Superchargers take about an hour to fully charge a Tesla EV.

Suppose you have been commissioned by an EV charging service provider to help them decide where to build their next charging station. In this assignment, you will design and implement a program for a charging station location selection problem. We use EV charging as our background scenario, but *you do not need to be an expert in EV or location selection to carry out this assignment.* Obviously, our assignment can only consider a much simplified version of the charging station location problem. In real scenarios, there are many factors that impact the choice of charging station locations, such as site availability, accessibility, etc.

[1]There are ongoing debates in the real benefits of EVs, and the purpose of this assignment is *not* to promote EVs.

We are given a *matrix of EV ownership numbers* that contains the numbers of EVs parked overnight at different locations, a list of *existing charging stations*, and a list of *candidate locations*. We study how to choose a location from the list of candidate locations to build the new charging station for maximising its benefits.

Below is a sample set of input data. The first line contains two positive integers `num_rows` and `num_cols` (separated by a space) representing the numbers of rows and columns of the matrix of EV ownership numbers. In the example below, `num_rows = 7` and `num_cols = 11`, meaning that there is a 7×11 matrix.

```
7 11
385 336 396 322 106 211 315 217 162 366 223
318 143 284 206 113 336 238 103 259 366 369
162 301 133 148 349 350 133 226 208 119 205
245 168 345 232 204 244 202 265 210 376 212
130 167 103 303 242 334 375 108 389 274 330
384 282 300 191 191 214 269 190 123 374 374
293 334 247 186 269 137 313 188 247 273 157
E 212 2.3 4.1
E 284 1.1 8.1
E 347 5.1 6.1
E 401 3.1 9.1
N 117 2.2 8.8
N 248 3.2 0.5
N 264 3.8 6.6
N 363 1.9 0.7
N 555 4.3 9.2
```

Starting from the second line, there are `num_rows` lines each with `num_cols` integers (separated by spaces) in the range of `[100, 999]`. The integer at the `i`-th row and `j`-th column represents the number of EVs parked overnight at a point[2] with coordinates `(i, j)`. In the example above:

- There are 385 EVs parked at `(0, 0)`, 336 EVs parked at `(0, 1)`, ..., and 223 EVs parked at `(0, 10)`.
- There are 293 EVs parked at `(6, 0)`, 334 EVs parked at `(6, 1)`, ..., and 157 EVs parked at `(6, 10)`.

The lines immediately after the matrix of EV ownership numbers represent a list of existing charging stations. Each such line represents an existing charging station with four values separated by spaces:

- The first is a character '`E`' indicating that this is a line of an existing charging station.
- The second is a unique (among the existing charging stations) three-digit integer ID of the existing charging station.
- The third and the fourth are real numbers representing the coordinates of the exiting charging station. They are in the ranges of `[0, num_rows-1]` and `[0, num_cols-1]`, respectively. No two charging stations share the exact same coordinates.

In the example above, the first existing charging station has an ID of `212` and is located at `(2.3, 4.1)`.

The lines after the list of existing charging stations represent a list of candidate locations for the new charging station. Each such line represents a candidate location with four values separated by spaces:

- The first is a character '`N`' indicating that this is a line of a candidate location.
- The second is a unique (among the candidate locations) three-digit integer ID of the candidate location.
- The third and the fourth are real numbers representing the coordinates of the candidate location. They are in the ranges of `[0, num_rows-1]` and `[0, num_cols-1]`, respectively. No two candidate locations share the exact same coordinates. The candidate locations do not share the exact same coordinates with the existing charging stations either.

In the example above, the first candidate location has an ID of `117` and is located at `(2.2, 8.8)`.

[2] A point here may represent a region such as a community or a suburb in a large map.

You may assume that the input is always correctly formatted, `1 <= num_rows <= 30`, `1 <= num_cols <= 20`, *there are between 1 and 99 existing charging stations, and there are between 1 and 99 candidate locations.* The existing charging stations and the candidate locations are sorted in ascending order of their IDs, respectively. No input validity checking is needed.

The tasks described below lead you to the desired program.

3 Your Task

3.1 Stage 1 - Read the Matrix of EV Ownership Numbers (Marks up to 5/20)

Write a program that reads the first line and and the matrix of EV ownership numbers from the standard input and stores the matrix into an array. Print out the total number of EVs (that is, the sum of all numbers in the matrix) and the point with the maximum number. If there is a tie, print out the point that has the smallest *Euclidean distance* to point `(0, 0)`. The Euclidean distance between two points $p_1 = (row_1, col_1)$ and $p_2 = (row_2, col_2)$ is calculated as:

$$\text{distance}(p_1, p_2) = \sqrt{(row_1 - row_2)^2 + (col_1 - col_2)^2} \tag{1}$$

If there is a further tie in the distance, print out the point with a smaller row coordinate.

The output of this stage given the above sample input should be:

```
Stage 1
==========
Total number of EVs: 19102
Maximum number of EVs: 396 at (00, 02)
```

Hint: A two-dimensional array to store the input matrix will suit the task nicely. Use `double` type for distance calculation and storage for best accuracy. Use `%02d` to print out the point coordinates. You may also read all input data before printing out for Stage 1.

3.2 Stage 2 - Read the Existing Charging Stations and Draw an Access Map (Marks up to 10/20)

Continue to read the existing charging stations from the standard input and store them into an array of `structs`. Then, assuming that people charge at their closest charging station, plot a *charging station access map* based on the locations of the charging stations as follows.

A charging station access map is a `num_rows x num_columns` grid of charging station distance values. For every point in the matrix of EV ownership numbers, the map visualises the Euclidean distance of the charging station closest (that is, having the smallest distance) to the point.

On the same sample input data, the additional output for this stage should be (there are no trailing spaces at the end of each line; same for Stage 3 output):

```
Stage 2
==========
E D D C C C C B B B C
E D C B B B C B A A B
E D C B A A B B A B B
E D C B A B C C B A A
E D C C B B B B B A B
E E D C C B A A B B C
F E E D C B A B C C D
```

Here, a distance value `d` is printed as a character `c = d + 'A'`. For example, for point `(2, 9)` (at the third line and last but second column), its distances to the four charging stations are: $\sqrt{(2.3-2)^2 + (4.1-9)^2} = 4.91$, $\sqrt{(1.1-2)^2 + (8.1-9)^2} = 1.27$, $\sqrt{(5.1-2)^2 + (6.1-9)^2} = 4.24$, and $\sqrt{(3.1-2)^2 + (9.1-9)^2} = 1.10$. The closest distance among these is $d = 1.10$, and `1.10 + 'A'` is printed as `'B'`.

3.3 Stage 3 - Read the First Candidate Location and Draw a New Access Map (Marks up to 15/20)

Next, read the first candidate location from the standard input and redraw the charging station access map, assuming that a new charging station has been built at the this location (in addition to the existing ones). On the same sample input data, the additional output for this stage should be:

```
Stage 3
==========
E D D C C C C B B B C
E D C B B B C B A A B
E D C B A A B B A A B
E D C B A B C B B A A
E D C C B B B B B A B
E E D C C B A A B B C
F E E D C B A B C C D
```

Here, point `(2, 9)`'s distance visualisation has been updated to `'A'`, as its distance to candidate location `(2.2, 8.8)` is 0.28 which is smaller than 1.10 as calculated above, and `0.28 + 'A'` is printed as `'A'`.

3.4 Stage 4 - Find the Most Beneficial Candidate Location (Marks up to 20/20)

Comparing the sample output of Stage 3 and Stage 4, we find that for points `(2, 9)` and `(3, 7)`, the printed distances have been changed from `'B'` to `'A'` and from `'C'` to `'B'`, respectively. This is because their distances to their respective closest charging stations will be reduced if a charging station is built at the first candidate location. For points `(1, 10)`, `(2, 8)`, and `(2, 10)`, their distances to their respective closest charging stations will also be reduced, even though the characters to visualise their distance values stay unchanged because of rounding. We sum up the number of EVs at these points (that is, `119+265+369+208+205=1166`), and we call this sum the *number of beneficiaries* of the first candidate location.

Now continue to read the rest of candidate locations from the standard input. For each candidate location (including the first one), calculate and print out its number of beneficiaries. At the end of this stage, your program should also print out the candidate location with the maximum number of beneficiaries. If there is a tie, print out the candidate location with the smallest ID among the tied ones.

On the same sample input data, the additional output for this stage should be (note a final newline character '`\n`' at the end of the output):

```
Stage 4
==========
Candidate #117 at (02.2, 08.8): 1166
Candidate #248 at (03.2, 00.5): 4776
Candidate #264 at (03.8, 06.6): 1595
Candidate #363 at (01.9, 00.7): 5209
Candidate #555 at (04.3, 09.2): 2541
Most beneficial candidate location: #363
```

Hint: Use `%04.1f` to print out the candidate locations' coordinates. When calculating the number of beneficiaries of a candidate location, all other candidate locations should be ignored, that is, there is only one new charging station to be built, not multiple.

You should be very proud of yourself when you have completed this stage – you have just implemented a classic algorithm named *max-inf* for facility location optimisation!

4 Submission and Assessment

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

Submitting your code. To submit your code, you will need to: (1) Log in to Canvas LMS subject site, (2) Navigate to "Assignment 2" in the "Assignments" page, (3) Click on "Load Assignment 2 in a new window", and (4) follow the instructions on the Gradescope "Assignment 2" page and click on the "Submit" link to make a submission. You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.* Submissions made after the deadline will be marked with late penalties as detailed at the end of this document. Do *not* submit after the deadline unless a late submission is intended. Two hidden tests will be run for marking. Results of these tests will be released after the marking is done.

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.

Testing on your own computer. 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 testing 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.

Testing on Grok. You may also choose to develop your solution on Grok in the Assignment 2 module. Your code on Grok will not be marked. Make sure to submit via Gradescope after you are satisfied with your solution.

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 **4:00 pm Friday 21st October 2022** will lose penalty marks at the rate of three marks per day or part day late. Late submissions after 4:00 pm Sunday 23rd October 2022 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/your-course/manage-your-course/exams-assessments-and-results`

And remember, *C programming is fun!*