The University of Melbourne
School of Computing and Information Systems
COMP20005 Engineering Computation
Semester 2, 2017
Assignment 1
Due: 4:00pm Wednesday 20th September 2017

1 Learning Outcomes

In this assignment you will demonstrate your understanding of loops and `if` statements by writing a program that sequentially processes a file of input data. You are also expected to make use of functions and arrays.

2 The Story...

VicRoads, the road and traffic authority in Victoria, has released a set of traffic accident records for the last five years[1]. This data set allows us to analyse traffic accidents based on a variety of factors such as location, time, weather condition, road condition, road users, etc. The data provide insights for improving road safety, such as whether to redesign road networks, relocate traffic cameras, adjust speed limits, etc.

Below is a sample list of traffic accident records, where the six columns represent unique record ID, accident location longitude, accident location latitude, accident date (`day/month/year`), accident time of day (`hour`), and accident day of week.

```
2693452 145.060689 -37.810373 26/5/12 11 Saturday
2693453 144.991172 -37.883156 6/6/12 15 Wednesday
2693454 145.009458 -37.826952 24/5/12 09 Thursday
2693455 145.134597 -37.841545 6/6/12 17 Wednesday
2693456 145.294599 -37.888597 22/5/12 16 Tuesday
2693458 145.008044 -37.783914 6/6/12 18 Wednesday
2693459 145.125164 -37.659316 6/6/12 15 Wednesday
2693460 145.211208 -37.982087 5/6/12 17 Tuesday
2693461 145.110783 -37.872920 6/6/12 16 Wednesday
2693462 144.960667 -37.827123 6/6/12 17 Wednesday
```

For example, the first line of the list represents traffic accident #2693452. The traffic accident happened at $\langle 145.060689, -37.810373 \rangle$, at 11 in the morning of Saturday, May 26, 2012.

3 Your Task

In this assignment, your task is to analyse the spatial and temporal distributions of traffic accidents. You will be given a list of traffic accident records like the one shown above. Particularly, each line of the list contains the information of a traffic accident separated by spaces (' '), including the unique ID (a 7-digit integer), the accident location longitude (a real number with 6 digits after the decimal point), the accident location latitude (a real number with 6 digits after the decimal point), the accident date (three integers in the format of `day/month/year`, between 1/1/2012 and 1/8/2017). the accident time of day (an integer between 0 and 23), and the accident day of week (a string in {"`Monday`", "`Tuesday`", "`Wednesday`", "`Thursday`", "`Friday`", "`Saturday`", "`Sunday`"}). *You may assume that the list will always be correctly formatted and contain at least 1 and at most 99 records.*

[1] `http://vicroadsopendata-vicroadsmaps.opendata.arcgis.com/datasets/crashes-last-five-years`

Your task is to write a program that reads the list and output statistics calculated on it. The assignment consists of the following four stages (Stage 4 is for a challenge and is optional).

3.1 Stage 1 - Processing the First Record (Marks up to 5/10)

You can complete this stage without using strings. However, you may also use strings if you are confident that you will be proceeding to Stage 3.

Write a program that reads the first line of the input data, and prints out for the first record: the unique ID, the longitude, the latitude, the date, the time, and the distance (in kilometres, 2 digits before and after decimal point) between the accident location and a reference point with a coordinate of $\langle 144.963123, -37.810592\rangle$[2].

Given the coordinates of two points $p_1 = \langle long_1, lat_1\rangle$ and $p_2 = \langle long_2, lat_2\rangle$, where $long_i$ and lat_i $(i = 1, 2)$ represent the longitude and latitude, the distance (in kilometres) between p_1 and p_2, represented by $\texttt{dist}(p_1, p_2)$, is calculated based on the *haversine formula* as follows.

$$\begin{aligned}
&\texttt{dist}(p_1,p_2) = 6371 \cdot angle\_distance, \text{where} \\
&angle\_distance = 2 \cdot \texttt{atan2}(\texttt{sqrt}(chord\_length), \texttt{sqrt}(1 - chord\_length)), \\
&chord\_length = \texttt{sin}^2(\texttt{toRadian}(lat_2 - lat_1)/2) + \\
&\qquad \texttt{cos}(\texttt{toRadian}(lat_1)) \cdot \texttt{cos}(\texttt{toRadian}(lat_2)) \cdot \texttt{sin}^2(\texttt{toRadian}(long_2 - long_1)/2)
\end{aligned} \tag{1}$$

In the equation above, $\texttt{toRadian}(x) = x \cdot (3.14159/180)$ is a function that converts a longitude or latitude coordinate x to its radian value. You need to implement this function and define the constants properly in your code. Note that you should *not* use the constant `M_PI` provided by the `math.h` library as it is not supported by the assignment submission server under the assignment compilation settings.

The functions `atan2()`, `sqrt()` `sin()`, and `cos()` are provided by the `math.h` library. If you use this library, make sure to add the "`-lm`" flag to the "`gcc`" command at compilation.

The output of this stage given the above sample input should be (where "`mac:`" is the command prompt):

```
mac: ./assmt1 < test0.txt
Stage 1
==========
Accident: #2693452
Location: <145.060689, -37.810373>
Date: 26/5/12
Time: 11
Distance to reference: 08.57
```

Here, 08.57 is the distance between the location of the first traffic accident, $\langle 145.060689, -37.810373\rangle$, and the reference point, $\langle 144.963123, -37.810592\rangle$, as calculated using the haversine formula above.

As this example illustrates, the best way to get data into your program is to save it in a text file (with a "`.txt`" extension, `jEdit` can do this), and then execute your program from the command line, feeding the data in via *input redirection* (using `<`).

In your program, you will still use the standard input functions such as `scanf()` *or* `getchar()` *to read the data. Input redirection will direct these functions to read the data from the file, instead of asking you to type it in by hand. This will be more convenient than typing out the test cases manually every time you test your program.* Our auto-testing system will feed input data into your submissions in this way as well. To simplify the assessment, your program should **not** print anything except for the data requested to be output (as shown in the output example).

Note that we will provide a sample test file "`test0.txt`" in LMS. You may download this file directly and use it to test your code before submission. This file is created under MacOS. Under Windows systems, the entire file may be displayed in one line if opened in the `Notepad` app. You do **not** need to edit this file to add line breaks. The '`\n`' characters are already contained in the file. They are not displayed properly but the `scanf()` and `getchar()`functions in `C` can still recognise them correctly.

[2]This is the coordinate for Melbourne Central.

3.2 Stage 2 - Processing the Rest of the Records (Marks up to 7/10)

You can complete this stage without using strings. However, you may also use strings if you are confident that you will be proceeding to Stage 3.

Now modify your program so that the distance to the reference point $\langle 144.963123, -37.810592\rangle$ for all of the input accident data are computed and visualised. You may assume that the distance values are within the range of $(0, 100)$. On the same sample input data, the additional output for this stage should be:

```
Stage 2
==========
Accident: #2693452, distance to reference: 08.57 |---------
Accident: #2693453, distance to reference: 08.44 |---------
Accident: #2693454, distance to reference: 04.46 |-----
Accident: #2693455, distance to reference: 15.45 |---------+------
Accident: #2693456, distance to reference: 30.37 |---------+---------+---------+-
Accident: #2693458, distance to reference: 04.94 |-----
Accident: #2693459, distance to reference: 22.05 |---------+---------+---
Accident: #2693460, distance to reference: 28.94 |---------+---------+---------
Accident: #2693461, distance to reference: 14.70 |---------+-----
Accident: #2693462, distance to reference: 01.85 |--
```

You need to work out how the visualisation works based on this example. Note that you should *write functions* to process this stage where appropriate.

3.3 Stage 3 - Overall Reporting (Marks up to 10/10)

To complete this stage you will need to use strings to store the accident days. If you have completed Stages 1 and 2 without strings, you should save a copy of that version of your program into a separate file, as an insurance policy that can be submitted again later if you are unable to get your Stage 3 solution operational.

The additional output from this stage is the total number of accident records, the day of week with the most accidents, and the corresponding number of accidents on that day of week. In the case of ties, the day earlier in the week should be output ("`Monday`" is considered the first day in a week and "`Sunday`" is considered the last day in a week).

```
Stage 3
==========
Number of accidents: 10
Day of week with the most accidents: Wednesday (6 accident(s))
```

Hint: You may need to use an array of strings and an array of integers to keep track of the accident count for each day. Wherever appropriate, code should be shared between the stages through the use of functions. In particular, there should *not* be long stretches of repeated code appearing in different places in your program. Further examples showing the full output that is required are provided on the LMS.

3.4 Stage 4 - For a Challenge (and Not for Submission)

For a challenge, try printing out the traffic accident records sorted in the ascending order of their distances to the reference point. *But please don't submit these programs.*

4 Submission and Assessment

This assignment is worth 10% 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 `assmt1.c` for assessment; detailed instructions on how to submit 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 made before the deadline will be marked.

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: ./assmt1 < test0.txt    /* Here '<' feeds the data from test0.txt into assmt1 */
```

Note that we are using the following command to compile your code on the submission server.

```
gcc -Wall -lm -std=c99 -o assmt1 assmt1.c
```

The flag "`-std=c99`" enables the compiler to use a more modern standard of the `C` language – `C99`. To ensure that your submission works properly on the submission server, you should use this command to compile your code on your local machine as well.

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 20th September 2017** will lose penalty marks at the rate of 1.5 marks per day or part day late. Late submissions after 4:00pm Friday 22nd September 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!*