The University of Melbourne
School of Computing and Information Systems
COMP20005 Engineering Computation
Semester 2, 2020
Assignment 1

Due: 4:00pm Wednesday 30th September 2020

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...

Contact tracing is an important measure to prevent a virus from spreading through the community. In dealing with the COVID-19 pandemic, many countries, including Australia, have taken various contact tracing measures. A core component in contact tracing is to track people's travel trajectories, from which people who have been in close contact with a new case can be identified quickly.

In this assignment, we are given a set of trajectory data that records people's locations at different times. The aim is to check whether there is anyone who has violated the 5-km from home travel zone limit, and whether there have been people who are in close contact for an extended period of time.

Below is a sample list of trajectory records of a certain day (the exact date is omitted for simplicity). Each line of the list contains the information of a location visited by a user as separated by spaces (' '), including:

- the visited location latitude (a real number with 6 digits after the decimal point);
- the visited location longitude (a real number with 6 digits after the decimal point);
- the user's home location latitude (a real number with 6 digits after the decimal point);
- the user's home location longitude (a real number with 6 digits after the decimal point);
- the start time of the visit (an integer between 1 and $24 \times 3600 = 86400$; this is the second from the start of the day, for example, $31144 = 8 \times 3600 + 39 \times 60 + 4$, that is, 31144 means 08:39:04);
- the end time of the visit (an integer between 1 and 86400, and greater than the start time of the visit);
- the user ID (a string with 8 digit or lowercase characters).

```
-37.817365 144.968783 -37.797329 144.965705 31144 32144 8oz05szh
-37.810009 144.962800 -37.797329 144.965705 33867 44877 8oz05szh
-37.809657 144.965221 -37.797329 144.965705 85646 85746 8oz05szh
-37.822464 144.968863 -37.810256 145.002215 41497 42511 jon362ft
-37.742099 144.964337 -37.837977 144.975683 00145 02159 kwhb16v7
-37.810019 144.962800 -37.837977 144.975683 43000 55079 kwhb16v7
-37.822464 144.968863 -37.837977 144.975683 60270 61284 kwhb16v7
-37.822464 144.968873 -37.837977 144.975683 61285 82281 kwhb16v7
-37.845590 144.971467 -37.773574 144.968853 52475 52591 wuucozwj
-37.817368 144.968786 -37.798875 144.943082 30144 65032 y4tqqn2s
```

For example, the first line of the list represents that User `#8oz05szh` visited `<-37.817365, 144.968783>` at time `31144` (08:39:04) and left at time `32144` (08:55:44). The user's home is at `<-37.797329, 144.965705>`.

You may assume that the list will always be correctly formatted and contain at least 2 and at most 99 records. The list is sorted by the user ID and then by the visit start time, in ascending order.

3 Your Task

Your task is to write a program that reads the list of trajectory records and outputs 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)

Write a program that reads the first line of the input data, and prints out for the first record: the user ID, the latitude and the longitude of the visited location, the start and end times of the visit, and the distance (in kilometres, 2 digits before and after decimal point) from the home location to the visited location.

Given the coordinates of two points $p_1 = \langle lat_1, long_1 \rangle$ and $p_2 = \langle lat_2, long_2 \rangle$, where lat_i and $long_i$ $(i = 1, 2)$ represent the latitude and the longitude, 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.

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

In the equation above, $\texttt{toRadian}(x) = x \cdot (3.14159/180)$ is a function that converts a latitude or longitude 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 system 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 on your own machine.

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

```
mac: ./program < test0.txt
Stage 1
==========
User: #8oz05szh
Visited location: <-37.817365, 144.968783>
Start time: 31144
End time: 32144
Distance from home: 02.24 km
```

Here, `02.24` is the distance between the visited location of the first record, `<-37.817365, 144.968783>`, and the home location, `<-37.797329, 144.965705>`, as calculated using the haversine formula above.

Hint: To ensure the result accuracy, use `double`'s for storing the coordinates and calculating the distance value. Use `%05.2f` for the distance output formatting. You may also read all input data before printing out for Stage 1.

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 marking, your program should **not** print anything except for the data requested to be output (as shown in the output example).

We will provide a sample test file "`test0.txt`" on Canvas. 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.

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

Now modify your program so that the distance from the home location to each visited location is computed and visualised. You may assume that the distance values are within the range of $(0, 15)$. On the same sample input data, the additional output for this stage should be:

```
Stage 2
==========
#8oz05szh, distance from home: 02.24 |---
#8oz05szh, distance from home: 01.43 |--
#8oz05szh, distance from home: 01.37 |--
#jon362ft, distance from home: 03.23 |----
#kwhb16v7, distance from home: 10.71 |----+----+-
#kwhb16v7, distance from home: 03.31 |----
#kwhb16v7, distance from home: 01.83 |--
#kwhb16v7, distance from home: 01.83 |--
#wuucozwj, distance from home: 08.01 |----+----
#y4tqqn2s, distance from home: 03.05 |----
```

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

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

In this stage, your task is to find out the *close-contact pairs*, which are location visit record pairs from different users with a distance smaller than 2 metres, and the visit times overlap for more than 15 minutes (that is, 900 seconds). The additional output from this stage is the total number of close-contact pairs, and the close-contact pair with the longest overlap time. You may assume that there will **not** be a tie on the longest overlap time.

```
Stage 3
==========
Number of close-contact pairs: 2
Longest overlap time: 01877 seconds
#8oz05szh @ <-37.810009, 144.962800>
#kwhb16v7 @ <-37.810019, 144.962800>

/* The alphabetically smaller ID should be printed first;
   theres is at least one close-contact pair in the input data;
   there is a final newline '\n' at the end of Stage 3 output. */
```

In the sample input, there are two close-contact pairs: (1) `#8oz05szh @ <-37.810009, 144.962800>` and `#kwhb16v7 @ <-37.810019, 144.962800>`, with a distance of `1.11` metres for `1877` seconds, and (2) `#8oz05szh @ <-37.817365, 144.968783>` and `#y4tqqn2s @ <-37.817368 144.968786>`, with a distance of `0.43` metres for `1000` seconds. The first pair is printed as its overlap time is longer.

Hint: You may need to use two arrays of `double`'s to keep track of the visited location coordinates, two arrays of `int`'s to keep track of the visit times, and an array of strings to keep track of the user IDs.

Wherever appropriate, code should be shared between 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 Canvas.

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

In Stage 3, you may have checked every pair of location visit records to find all the close-contact pairs. Given n location visit records, there are n^2 pairs to be checked, many of which are non-close-contact pairs. For a challenge, can you think of a way to reduce the number of pairs to be checked without missing any close-contact pairs? *But no need to submit these programs.*

4 Submission and Assessment

This assignment is worth 10% of the final mark. A detailed marking scheme will be provided on Canvas.

You need to submit your code in Grok Learning (`https://groklearning.com`) for assessment. Submission will **not** be done via Canvas. Instead, you will need to:

1. Log in to Grok Learning using your student login details.
2. Navigate to the Assignment 1 module of our subject COMP20005 2020 S2: Engineering Computation.
3. Write your code in the `program.c` tab window.
4. Compile your code by clicking on the `Compile` button.
5. Once the compilation is successful, click on the `Mark` button to submit your code. (You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.*)
6. Two sample tests will be run automatically after you make a submission. Make sure that your submission passes these sample tests.
7. Two hidden tests will be run for marking purpose. Results of these tests will be released after our 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.

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 system (we name the source code file `program.c`).

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

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.

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 **4:00pm Wednesday 30th September 2020** will lose penalty marks at the rate of 1.5 marks per day or part day late. Late submissions after 4:00pm Friday 2nd October 2020 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 (HRP) 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/student-support/coronavirus/exams-special-consideration-and-WAM`

And remember, *C programming is fun!*