School of Electrical Engineering and Computing
COMP2240/COMP6240 - Operating Systems
Assignment 2 (15%)

Submit using Blackboard by **23:59, 18th October (Sunday), 2020**

Problem 1: Sharing the Bridge

A new single lane bridge is constructed to connect the North Island of New Zealand to the South Island of New Zealand. Farmers from each island use the bridge to deliver produce to the other island, return back to their island and this is repeated indefinitely. It takes a farmer carrying produce 20 steps to cross the bridge. Once a farmer (say a North Island farmer identified as `N_Farmer1`) crosses the bridge (from North Island to South Island) he just attempts to cross the bridge in the opposite direction (from South Island to North Island) and so on. Note that the ID of the farmer does NOT change.

The bridge can become deadlocked if a northbound and a southbound farmer are on the bridge at the same time (New Zealand farmers are stubborn and will not back up). The bridge has a large neon sign above it indicating the number of farmers that have crossed it in either direction. The neon sign counts multiple crossing by the same farmer.

Using **semaphores**, design and implement an algorithm that prevents deadlock. Use **threads** to simulate **multiple/concurrent** farmers and assume that the stream of farmers are constantly attempting to use the bridge from either direction. Your program should input parameters at runtime to initialise the number of farmers from each direction. For example `[N=5, S=5]` would indicate a constant stream of 5 farmers from each direction wanting to use the bridge. You also make sure that the solution is starvation-free (the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa should not occur).

Add 200 ms delay between every 5 steps of the farmers take.

Sample Input/output for Problem 1.

The input will be as follows:

```
N=2, S=2
```

The input indicates the program is initialized with 2 farmers from each direction who will constantly attempt to use the bridge to go from one island to other.

The (**partial**) output from one execution is as follows:

```
N_Farmer1: Wating for bridge. Going towards South
S_Farmer2: Wating for bridge. Going towards North
S_Farmer1: Wating for bridge. Going towards North
N_Farmer2: Wating for bridge. Going towards South
N_Farmer1: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 10.
N_Farmer1: Crossing bridge Step 15.
N_Farmer1: Across the bridge.
NEON = 1
S_Farmer2: Crossing bridge Step 5.
N_Farmer1: Wating for bridge. Going towards North
S_Farmer2: Crossing bridge Step 10.
S_Farmer2: Crossing bridge Step 15.
S_Farmer2: Across the bridge.
NEON = 2
N_Farmer2: Crossing bridge Step 5.
N_Farmer2: Crossing bridge Step 10.
N_Farmer2: Crossing bridge Step 15.
S_Farmer2: Wating for bridge. Going towards South
N_Farmer2: Across the bridge.
NEON = 3
S_Farmer1: Crossing bridge Step 5.
N_Farmer2: Wating for bridge. Going towards North
S_Farmer1: Crossing bridge Step 10.
S_Farmer1: Crossing bridge Step 15.
S_Farmer1: Across the bridge.
NEON = 4
N_Farmer1: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 10.
S_Farmer1: Wating for bridge. Going towards South
N_Farmer1: Crossing bridge Step 15.
N_Farmer1: Across the bridge.
NEON = 5
…
…
…
```

NOTE: For the same input the output may look somewhat different from run to run.

Problem 2: Covid-safe restaurant

Covid-19 has changed everything this year and restaurants are not any exception. In order to comply with the public health orders, a restaurant has developed a Covid-19 safety plan. The restaurant does not offer any take away service. In order to satisfy the "*one customer per 4 square metres of space*" rule it can allow **only five customers** to eat in at a particular time. The manager's Covid-19 plan is as follows: (i) the customers will be served in FIFO order (ii) if a customer arrives when there is an empty seat (out of five usable seats), then the customer can immediately take a seat. (iii) If all the five seats become occupied (i.e. there are five customers enjoying their meals at any instant) then all the arriving customers have to wait for the entire party (all current customers) to leave and perform a cleaning before the waiting customers can get their seats. The cleaning takes 5 minutes. Note that a customer may arrive to the restaurant at any time and may take a different time to finish his/her meal.

Using **semaphores** design and implement an algorithm that manages the customers entering and leaving the restaurant in line with manager's rules. Use **threads** to simulate multiple/**concurrent** customers. Your solution must be fair and starvation free, i.e., the customers will be served in the order of their arrival time and no customer should be waiting for indefinite time. Assume no time is wasted in taking seat, serving/starting eating meals and leaving the parlour.

The input will be as follows:

```
0 C1 5
0 C2 7
0 C3 8
2 C4 5
3 C5 5
4 C6 3
7 C7 5
14 C8 5
END
```

Where each line, except the last, contains information regarding a customer
Arrival-time Customer-ID eating-time
Input ends with a line containing the word END.

You can assume that in the input the customers will be sorted in order of their arrival times.

The output should be as follows:

```
Customer    arrives      Seats    Leaves
C1              0          0         5
C2              0          0         7
C3              0          0         8
C4              2          2         7
C5              3          3         8
C6              4         13        16
C7              7         13        18
C8             14         14        19
```

Output shows information of each customer in a separate line. Each line contains Customer-ID, Arrival-time, time when the customer seats in the ice-cream parlour and time when the customer leaves the parlour.

Problem 3 : Covid-safe restaurant using monitor

You will need to implement a solution for Problem 2 (Covid-safe restaurant) using monitor.

Using **monitor**, design and implement an algorithm that manages the customers entering and leaving the restaurant in line with manager's rules. Use **threads** to simulate multiple/**concurrent** customers. Your solution must be fair and starvation free, i.e., the customers will be served in the order of their arrival time and no customer should be waiting for indefinite time. Assume no time is wasted in taking seat, serving/starting eating meals and leaving the parlour.

The sample input/output will be same as show in Problem 2.

Problem 4: Share the Bridge in Pair [ONLY for COMP6240 students]

The single lane bride in Problem 1 has been upgraded to two lanes allowing two farmers to travel simultaneously. In order to share the toll, farmers cross the bridge only in pairs – a farmer will never cross the bridge alone. In order to avoid possible conflicts, either two North Island farmers or two South Island farmers can cross the bridge in the same direction. Farmers from each island use the bridge in pairs to deliver produce to the other island. As before, it takes a farmer 20 steps to cross the bridge. Once two farmers (say two North Island farmers identified as `N_Farmer3` and `N_Farmer4`) cross the bridge (from North Island to South Island) they will attempt to cross the bridge in the opposite direction (from South Island to North Island), however, the farmers may pair with other North Island farmers (e.g. `N_Farmer3` may pair up with `N_Farmer2` if `N_Farmer2` was waiting in the South Island for the bridge/pair).

The bridge can become deadlocked if two northbound and two southbound farmers are on the bridge at the same time as the stubborn farmers will not back up. The large neon sign indicates the number of farmers that have crossed it in either direction and will count multiple crossing by the same farmer.

Using **semaphores**, design and implement an algorithm that prevents deadlock. Use one **thread** to represent each farmer and simulate **multiple/concurrent** farmers and assume that the stream of farmers attempt to use the bridge in pairs from both directions. Your program should input parameters at runtime to initialise the number of farmers from each direction. For example, the input `[N=5, S=5]` would indicate a group of 5 farmers from each direction want to use the bridge. You also make sure that the solution is starvation-free (i.e. the situation in which northbound farmers prevent southbound farmers from using the bridge, or vice versa should not occur).

Add 200 ms delay between every 5 steps of the farmers take.

Sample Input/output for Problem 4.

The input will be as follows:

```
N=2, S=2
```

The input indicates the program is initialized with 2 farmers from each direction who will constantly attempt to use the bridge to go from one island to other.

The output (partial) from one execution is as follows:

```
N_Farmer1: Wating for bridge. Going towards South
S_Farmer2: Wating for bridge. Going towards North
S_Farmer1: Wating for bridge. Going towards North
N_Farmer2: Wating for bridge. Going towards South
S_Farmer1: Crossing bridge Step 5.
S_Farmer2: Crossing bridge Step 5.
S_Farmer2: Crossing bridge Step 10.
S_Farmer1: Crossing bridge Step 10.
S_Farmer2: Crossing bridge Step 15.
S_Farmer2: Across the bridge.
NEON = 1
S_Farmer1: Crossing bridge Step 15.
S_Farmer1: Across the bridge.
NEON = 2
N_Farmer2: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 5.
N_Farmer1: Crossing bridge Step 10.
N_Farmer1: Crossing bridge Step 15.
N_Farmer1: Across the bridge.
NEON = 3
N_Farmer2: Crossing bridge Step 10.
N_Farmer2: Crossing bridge Step 15.
N_Farmer2: Across the bridge.
NEON = 4
…
…
```

NOTE: For the same input the output may look somewhat different from run to run.

Programming Language:

The programming language is Java, versioned as per the University Lab Environment (**currently a subversion of Java 1.8**). You may only use standard Java libraries as part of your submission.

If you wish to use any language other than the preferred programming language, you must first notify (and justify this to) the course demonstrator (Dan).

User Interface:

The output should be printed to the console, and strictly following the output samples shown above (also given in the assignment package as separate files). You will lose marks if your program does not conform with the input/output formats.

Input and Output:

Your program will accept data from an input file of name specified as a command line argument. The sample files `P1-1in.txt` and `P2-1in.txt` (containing inputs for Problem 1 and 2/3 respectively) are provided to demonstrate the required input file format.

Your submission will be tested with the above data and will also be tested with other input files.
Your program should output to standard output (*this means output to the Console*). Output should be strictly in the given format.

The sample files `P1-1out.txt` and `P2-1out.txt` (containing output for `P1-1in.txt` and `P2-1in.txt` respectively) are provided to demonstrate the required output (and input) format which **must be strictly maintained. If output is not generated in the required format then your program will be considered incorrect.**

Deliverable:

1. Your submission will contain **three folders** (each containing source code for one problem), **documentation** (below) and a **readme.txt** (containing any special instructions required to compile and run the source code) in the root of the submission. Name the folders **1**, **2** and **3** for Problem 1, 2 and 3, respectively. Zip these files and folder up into an archive called **`c12345678.zip`** (where **`c12345678`** is your student number)- do not submit a `.rar` or a `.7z` etc.
2. Your main classes for different problems should be **`P1.java`**, **`P2.java`** and **`P3.java`** and your program will compile with the command line **`javac P1.java`** (for P1.java) and your program will be executed by running **`java P1 input.txt`** (where `input.txt` is a relative path name to any variety of filename, both with or without any sort of extension – do not hard code anything!). Obviously this will hold true for P2 and P3 with appropriate changes for the main source file.
3. Brief 1 page (A4) review (report = 10%) of the how you tested your programs to ensure they enforced mutual exclusion and are deadlock and starvation free.

NOTE: Assignments submitted after the deadline **(11:59 pm Sunday 18th October 2020)** will have the maximum marks available reduced by 10% per 24 hours.

Mark Distribution:

Mark distribution can be found in the assignment feedback document (Assign2Feedback2240.pdf/ Assign2Feedback6240.pdf).

Threads:

You will use threads to simulate the concurrent elements. To get you started, have a look at the following tutorials:

For Java: http://docs.oracle.com/javase/tutorial/essential/concurrency/
For C/C++: https://computing.llnl.gov/tutorials/pthreads/