University of Exeter

College of Engineering, Mathematics

and Physical Sciences

ECM2433

The C Family

Continuous Assessment

Date Set:	28 th February 2022
Date Due:	28 th March 2022
Return Date:	2 nd May 2022

This CA comprises 60% of the overall module assessment.

This is an **individual** exercise and your attention is drawn to the College and University guidelines on collaboration and plagiarism, which are available from the College website.

This is the only coursework for this module and tests your skills and understanding of programming in C.

Please Turn Over

Problem Statement

You are working for a company that installs temporary traffic lights for roadworks. Your task is to design and build a simulation of a simple installation, so that you can test the performance of the algorithm that controls the lights. Figure 1 shows two views of the sort of temporary lights you will be simulating. There are two sets of traffic lights shown, protecting a section of road that is not wide enough for two vehicles to pass each other in opposite directions. Figure 2 shows a simplified diagram of the traffic light installation you are going to simulate.

Your task is in two parts:

1. Create a simulator for this system.
2. Run the simulator multiple times to establish how it operates under different traffic conditions and traffic light timings.

Figure 1: Two images of a real-life set of temporary traffic lights, protecting a single-track section of the road.

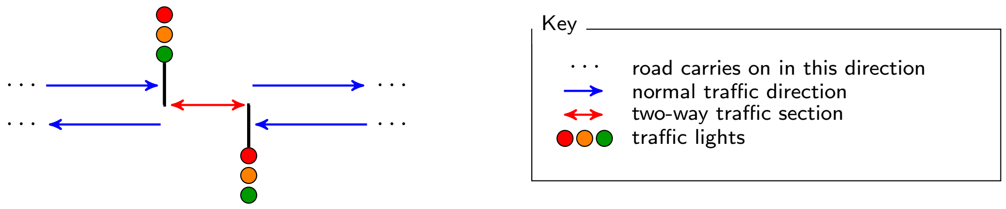

Figure 2: A diagram of the setup shown in figure 1, reduced to the essentials required for the simulation.

Simulator

Your simulator will make use of random numbers. This means that a single run of a simulation might be exceptional in some way, just because of the sequence in which the random numbers are generated. So in order to assess the value of a particular simulation you must run it multiple times (i.e. using different sequences of random numbers, but with the same setup) and look at the average results over all of them.

In your simulator, the rates at which vehicles arrive from the left and from the right will be controlled by random numbers. The other factor controlling the outcome of the simulation is the length of time the traffic lights are green in each direction. This period is usually the same in each direction, but might be different. Hence a simulation is controlled by four parameters.

The simulator is to be developed in two parts:

1. A function, called `runOneSimulation`, which accepts the four parameter values as inputs, runs a single simulation according to those parameters, and then returns the results. This function, and any other functions you write that it calls, must be written in ANSI standard C.
2. A program called `runSimulations` that accepts the four parameter values from the command-line at runtime, runs 100 single simulations based on those parameter values, and then outputs

the average results across the 100. This program may be written in *either* ANSI standard C *or* C++11 standard C++.

Single simulation: `runOneSimulation`

Each simulation is an iterative process, where each iteration represents one "time" step. It does not matter how long this period is in actual time units (e.g. seconds). At any instant, one set of traffic lights shows red and the other green. They cannot both be red or both be green. We will not simulate the effects of the amber lights. Within one iteration/time period, *either* the lights change *or* each of the following happens, in this order:

- zero or one vehicle arrives from the left and joins the left-hand queue,
- zero or one vehicle arrives from the right and joins the right-hand queue,
- zero or one vehicle passes through the protected road section between the two sets of traffic lights, either from left to right, or right to left, depending on which of the sets of lights is green.

After 500 iterations no more vehicles arrive, but the lights keep changing and vehicles pass through the protected section. The simulation ends when there are no more vehicles remaining in either queue.

At the end of a simulation, we want to know, for each direction separately (i.e. for vehicles that passed through the traffic lights from left to right, and then separately for those that travelled from right to left):

(a) The number of vehicles that passed through the traffic lights.
(b) The average waiting time.
(c) The maximum waiting time.
(d) The time taken to clear the queue once vehicles have stopped arriving.

Whether a vehicle arrives to join a queue in any one timestep is determined by a random number. In your experiments, you might consider two different simulations: (1) the probability of vehicles arriving is the same in each direction, and (2) the probability of vehicles arriving from one direction is higher than the other.

Multiple simulations: `runSimulations`

Once you have created the code for one simulation, as described above, you will run that code multiple times with different parameter settings to investigate the effects of different values of the parameters. For a single set of the four parameter values, the simulation must be run 100 times, and the average results printed out to the *stdout* stream. The output should look something like this:

```
Parameter values:
   from left:
      traffic arrival rate: XXX
      traffic light period: XXX
   from right:
      traffic arrival rate: XXX
      traffic light period: XXX
Results (averaged over 100 runs):
   from left:
      number of vehicles:   XXX
      average waiting time: XXX
      maximum waiting time: XXX
      clearance time:       XXX
   from right:
      number of vehicles:   XXX
      average waiting time: XXX
      maximum waiting time: XXX
      clearance time:       XXX
```

where the "`XXX`" are replaced with the relevant numbers.

Report

As well as the simulator, you must submit a report (maximum 4 pages of A4). This must include the following:

- A description of the design decisions you have made when developing your program and any assumptions you have made.
- A description of an experiment you have performed using your simulator, and the results you obtained, including a discussion about the meaning of those results.
- Example output from at least one run of your code.

If your submitted code does not work fully as expected, then describe the details in the report, including any testing you have performed to narrow down what is causing the problem.

Deliverables

The deliverables for this coursework comprise the source code for your simulator, instructions for compiling and linking it into an executable, and your report, zipped together and submitted via EBART. Your submission must contain the following:

- The source code for the `runOneSimulation` function, which must be written in well-structured, ANSI standard C.
- The source code for the `runSimulations` program, which must be written in well-structured, ANSI standard C or C++11 standard C++.
- A Linux shell script for compiling and linking your code, called `compileSim` (and nothing else), and must compile your code to an executable called `runSimulations` (and nothing else).
- The report, as detailed in the previous section.

Your program must compile, link and execute on one of the two Linux servers we are using on this module (`emps-ugcs1` and `emps-ugcs2`) and will be tested on one of those servers.

Submission must be made by 12pm (noon) on the date indicated on the front page of this document.

Marking Scheme

Criteria	**Marks**
Program basics *To what extent is your code well-constructed, well-formatted, and to the appropriate standard (ANSI/C++11)? Does your program compile and link without errors and warnings? To what extent does it trap and report run-time errors? To what extent does your program correctly read in the command line parameters?*	40
Function of the simulator *To what extent does the program make a suitable representations of the entities in the system? (Top marks will only be awarded if the queues are implemented as linked lists.) To what extent does the program correctly and efficiently perform the simulations and output the results?*	40
Report *To what extent does the report clearly and concisely describe your design decisions and assumptions? To what extent does it describe a plausible experiment, its outputs and the meaning of those outputs? Does the report include the specified example output? If the code does not work fully, to what extent have the errors been investigated and described in the report? Are the spelling, grammar, language and presentation of the report clear, formal and professional and appropriate to the intended audience?*	20
Total	100

Penalties

You will be penalised 10 marks if the executable code is not called `runSimulations`, or the Linux shell script for compiling and linking it is not called `compileSim` (or has not been included).

If your report exceeds 4 pages, only the first 4 pages will be marked.