1 Design Decisions

I developed the program to run the simulation in 2 separate modules, runSimulations and runOneSimulation. The actual program that is being called and run is called runSimulations. In this module the majority of the setup is done, with the user inputting the parameters and the structures to store the results of the simulations being created. The results are also processed and outputted within this module. It also calls the function runOneSimulation from runOneSimulation.c. and runOneSimulation. The actual program that is being called and run is called runSimulations. In this module the majority of the setup is done, with the user inputting the parameters and the structures to store the results of the simulations being created. The results are also processed and outputted within this module. It also calls the function runOneSimulation from runOneSimulation.c.

The runOneSimulation module is where an individual simulation is created and run, with the results returned to runSimulations. The module contains the function runOneSimulation which is responsible for setting up and running a single simulation. The module also contains all the necessary functions and structures to be able to do so. The majority of these are not available to runSimulations as runSimulations does not need to access these functions or structures to work properly.

I separated the code into these separate modules in order to increase the compartmentalization of the code. This makes the code more modular and secure as well as easier to read, debug, modify or extend.I also tried to make the code more modular by making lots of subsidiary functions to create necessary structures or tasks such as adding a car to the queue. I also did this with the processing of the data so that if this simulation wanted to focus on different results it would be easy to do so. In doing so the individual functions can be later modified or extended rather than having to sift through one very large function and try to figure out what exactly does what. While it does add more function calls and requires the use of more pointers, adding complexity, and an increased risk of memory leaking. Because it is a very small program and runs near instantly the additional function calls do not cause a noticeable slowdown of the simulations, this is even when running thousands of individual simulations. Therefore I thought that this was a good trade off to make the code more modular.

I implemented the representation of the cars waiting at either end of the traffic lights as a linked list implemented as a queue with each node representing a car. This is because unlike arrays it means that the memory allocated for the queue does not have to be allocated as contiguous memory. I also do not know how many cars will be added for each trial and this allows the necessary memory to be dynamically allocated at runtime rather than allocate an array at the start.

The implementation also doesn't require any searching through the linked list to be done which would be O(n). This is because the head and tail of the queue are stored withing the queue which are the only nodes that need to be accessed. This is because cars are only ever removed from the front of the queue

or added to the back of the queue. Accessing these nodes is an O(1) operation. This means that in this specific scenario with this implementation there is no trade off in the time complexity of finding nodes compared to an array.

The only other data the queue stores is the size of the queue, which is updated every time a node is added or removed. While this involves a lot of updating memory it means that the entire linked list does not need to be traversed to know its size, making the program more efficient. This becomes more relevant the longer the queue (the higher the rates of arrival) or if a longer time period than 500 is observed.

I created structures for most things rather than instantiate a lot of loose variables. For example the results that need to be updated. Instead of passing 8 pointers to runOneSimulation only one pointer to a results structure is passed. I also tried to limit the amount of memory reallocation that had to happen by resetting the values of some structures, to reduce possible memory leakages.

I did however create new structures such as queues, cars and greenlights for each individual simulation. I did this because it made the program a lot easier to debug and will make it much easier to extend if for example a third traffic light was added a third queue could be added and the greenlight structure slightly modified. It also ensures that each simulation starts in the exact same state.

I seeded and set up the environment for the random number generator, used to determine whether cars are allowed to pass through in the runSimulations function and pass the pointer to its state through to each call of runOneSimulation. This is because I initially seeded it inside runOneSimulation using time(0). However I found that often time(0) would be the same value in different simulations and therefore the random number generator would be identical for different individual simulations, creating duplicate simulations. I also used the gsl library and its random number generators because while currently the number of simulations is very low it could easily be greatly increased and therefore I wanted a higher quality of random number generation than using rand().

I output the results to a file called results.txt using the stdout stream, as per the coursework specifications. I output the results to a file because I found it much easier to store the results of trials, rather than reading through the terminal and needing to record the results before quitting the terminal. I used a .txt file which is different than the examples in the slides and this is because I did not feel comfortable using a .lis file which I do not know much about.

I also used ints for most of the numbers within the structures. This is to keep the program simple and consistent. With the simulation as it is there is no need for numbers larger than what the int can store. For many values an int is overkill with only an unsigned short int being necessary, however the application does not take up a lot of memory and does not need this optimization. The simplicity of consistently using ints is worth the trade off. If the program were expanded to include simulations of larger sizes this is something to consider.

Finally I assumed that the time for a car to arrive and the time for a car to drive through the protected area and the time to change the colour of the traffic lights is the same. I also didn't involve the use of amber in the simulation. This obviously simplifies the simulation a lot and makes it less applicable to real life,

but it makes the simulation a lot simpler to code.

2 Experiment and Results

I wanted to test the effects of changing each parameter. I realized that changing the rates of arrival of either side only affects that respective side. Increasing the rate of arrival only increases every recorded statistic of the queue that has an increased rate of arrival. I also noticed that when using low rates of arrival for either side changing the time the traffic light changed colours did not make a large difference. This is because there are very few vehicles arriving and the queue will not be large so it is likely that every time a traffic light turns green that side's queue can completely empty before the traffic light changes colour.

Therefore I decided to see how the different times that each traffic light stays green for affects the statistics. I used an arrival rate of 0.6, as this is high enough to see results but not too high where waiting times are too high to properly analyze. I used traffic light ranges from 5-50 because I felt that anything more or less presents an unrealistic scenario compared to real life roadworks, as well as provide an unoptimal solution for the company that is tasking me with the simulation. TLTX is traffic light time X. Where x is the side. WT is waiting time. CT is clearance time. Below is the results of a trial where I kept one light at 20. I would like to show all the results but there is not enough space.

TLTL	TLTR	# Vehicles (L)	# Vehicles (R)	avg WT (L)	avg WT (R)	max WT(L)	max WT(R)	CT (L)	CT (R)
20	10	282	280	4	208	15	409	3	404
20	15	284	283	22	119	48	215	36	210
20	20	287	286	58	60	117	119	110	112
20	25	287	288	93	32	188	64	181	52
20	30	288	288	129	16	262	36	254	17

I found that the lowest average waiting time occurs when the traffic lights are equal to one another. The lowest average waiting time totals at 118, whereas other variations have larger average waiting times. The larger the difference in the time that the two traffic lights are green, the larger the average waiting time. The size of the average waiting time is also increased by the proportion that the difference is in comparison with the time that each traffic light is green. e.g. the traffic light times of 20 and 10 will have a longer average waiting time than 20 and 30. The same trend is seen in maximum waiting time and clearance time. Below is the average data of each side, which demonstrates the above trends.

TLTL	TLTR	total avg waiting time	avg max waiting time	total clearance time
20	10	212	215	407
20	15	131	141	246
20	20	118	119	222
20	25	125	127	233
20	30	145	149	271

Because equal traffic light values yield the lowest waiting and clearance times I wanted to see what equal traffic light times yield the lowest waiting and clearance times. I plotted the averages below because they are near identical values for each side, they only differ slightly because of the random number generator.

TLTL	TLTR	avg waiting time	avg max waiting time	total clearance time
5	5	52	107	103
10	10	55	109	105
20	20	58	114	107
30	30	63	125	114

The lower the time that the traffic lights are green the lower every statistic measured is. This is consistent for different rates of arrival. I find this surprising because changing colours takes a timestep and slows things down. This finding is however slightly unrealistic because it is assuming that the time to change lights colour is the same time as it takes for a car to drive through the protected area. In reality the most useful value is likely higher as it allows a number of cars through before switching colours. The experiment also keeps the rate of arrival from each side the same, which is unlikely to be true as for example during rush hour in the morning one side may be busier than the other, which should be considered when deciding traffic light values.

3 Example Output

Parameter values:
    left traffic arrival rate: 0.800000
    left traffic light period: 4
    right traffic arrival rate: 0.800000
    right traffic light period: 4
Results (averaged over 100 runs):
    from left:
        number of vehicles: 320
        average waiting time: 149
        maximum waiting time: 301
        clearance time: 299
    from right:
        number of vehicles: 319
        average waiting time: 151
        maximum waiting time: 303
        clearance time: 301