ECM2433 CA Traffic Simulation Report

**Design Decisions**

The most immediate design decision when devising how the solution should be programmed was deciding how the Traffic should be modelled. I decided to use structs to represent the two traffic sides, as left queue and right queue implemented as linked lists. 

*Figure Implementation of Queue Struct*

This allowed me to model individual vehicles as structs of type Car, which then contain variables for result gathering. To model the traffic lights I used one variable currentGreen, which when 0 would indicate the left traffic could pass and right could not, when 1 right traffic could pass and left traffic could not.

When initially programming my solution, the function runOneSimulation() would repeatedly call one of two functions to process each iteration. These were addCarIterate() and noCarIterate(). These functions both then would use conditionals to call different functions based on which actions were occurring that iteration (adding a car to a queue and changing the lights are examples of these actions). To use the variables that I had defined outside of the scope of these functions every variable had to be a pointer, where the pointer would then be passed around the functions. Using pointers for all of the variables lead to many segmentation faults, but they were solvable up until I began to implement result gathering to display the outputs of a simulation. As I could seemingly not overcome the issue I had after trying to work through it I decided it would be better to remove the pointers and work instead with simple variables. To still be able to change the values of the variables when not accessing them through a pointer, I had to move all of the code into one function. This allows for all the variables to be in scope of the function operating on them, except the four parameter variables which are constants to be set before a simulation run. 

*Figure removed function addCarIterate() took every variable as a pointer to update its values*

Using only one function has removed the need for pointers in almost all cases, it has also cut down on some functions that had to take ridiculous numbers of pointers as arguments to be able to update them all. It also cut down on some repeated code, however obviously in replacing each function call with the full code of the function will lead to some redundant code. Due to this the code originally from the function newCar() has been repeated twice and the code from the function vehiclePass() has been repeated four times. These are the only parts of code that had to be repeated when written into only one function.



*Figure removed function vehiclePass() was not a seperate function in final iteration*

I collected the results in a struct Results, which allows for more result parameters and such to be added easily, were the program to be further expanded upon. These results are returned from the runOneSimulation() function, so that the results of a single simulation can be taken and evaluated alone. This proved useful for error checking during programming as being able to see results for just one run gave me insight into where erroneous code might be.

In my program I decided that almost all variables would be of type int, with the only exceptions being how the results and Traffic Queues were modelled. This choice was made as it allows me to avoid having to use type conversions on any of the variables, which would have otherwise been necessary when comparing variables through conditionals.

**Experiment**

Using my simulation I ran an experiment to see what the greatest range of car totals and of average waiting time would be, when the program was given the same variables to run on each time. The program ran the parameters 10,10,10,10 successfully 3 times. Ignoring obviously erroneous results, the results are listed below.

|  |  |  |  |
| --- | --- | --- | --- |
| Results when given parameters 10,10,10 and 10 | | | |
| L Total Cars | R Total Cars | L Waiting Time | R Waiting Time |
| 224 | 235 | 200 | 300 |
| 168 | 205 | 84 | 183 |
| 405 | 251 | 391 | 213 |

Although it is still possible that the average waiting times for the cars are erroneous (as the avg wait times tend to be a much higher number than expected), the results are all lower than the total amount of cars added so could be correct. If assumed to be correct we can see that when the number of cars goes up, the average waiting time increases at a rate such that it begins near 0.5 when 168 cars are given, reaching near 1 at ~200 cars and staying near 1 through to 405 cars. From this experiment we can also gather that there is no relationship between the total cars on the left and right sides, nor any relationship between the left and right waiting times. There is however strong positive correlation between the total cars and average waiting times of each queue.

**Output example**

This successful run of my code was used in my experiment.



*Figure 4 Example output of runSimulations without error*