# Module F

## Goals-

Design, write, compile, and test a program that uses a simple linked structure

This module builds on the previous module. You will receive feedback as soon as possible. **Save a copy of the previous module** so you have a known place to start in case of catastrophe! Design your changes. Review the design. Then, and only then, start coding!

In Module E you implemented a simulation of a traffic intersection. You could count the number of cars arriving and departing, and how many got stuck at the light. What if you want to simulate something about individual cars? Such as average wait time? Or you want to add multiple traffic lanes and model drivers switching lanes? You add a queue for each traffic lane, i.e. one for eastbound, southbound, and so on. A queue will allow you to track information per vehicle.

First, you need a queue. A queue is a first in first out (FIFO) structure. You will need a pointer to the front and to the back of the queue. You are not required to implement any other features of the formal Queue data structure. Be careful when you research this. You could make your program more difficult.

You will create a doubly linked Queuenode. You will need a front pointer and a back (or rear) pointer to the node on either end. You will need to only add an element at the back node and to only take off an element from the front node. You should have functions:

void add(Type_of_Data Value)
Type_of_Data remove()

Type_of_Data will be int. For each "car" put in the queue you will use the timestamp (i.e. which iteration of the loop you are on). For the number of cars waiting you can continue to use the data member, or you could add a size function to your queue. What is the disadvantage to adding a size function? Use the data member so you need to change less code. Your functions would still need to return the values to update the cars waiting data member.

ANSWER: A function call has overhead and is less efficient. For the few lanes and small amount of traffic it's probably not a problem. But if you add many lanes and many intersections it might be a concern. Again, keep it simple and don't change the code unless you are required to change it.

There will be a queue for each direction of travel. Your function to calculate the number of cars arriving would calculate the number of cars and add them to the back of the queue. Each entry would be the timestamp. Your function to calculate the number of cars departing would just call remove for each one. So:

How do you calculate the average wait time? For each traffic lane you count the number of cars. For each traffic lane you total (sum) the time each car was waiting. How do you do that? The car entry is the timestamp it was put in the queue. You subtract that from the current

timestamp. Take that difference and add it to your sum. This will require at least one new data member.

It is iteration 42 and your function determines 7 cars arrive. You would use a loop to push that value 42 into the queue 7 times.

It is iteration 43 and your function determines 10 cars get through the intersection. You use a loop to remove the front 10 values, say: 37, 37, 37, 37, 37, 39, 39, 39, 40, 40. The first car waited 43-39 or 4 cycles of the light.

At the end of the simulation run, you divide the total wait time by the number of cars and voila! You have the average.

NOTE: De-allocate memory as appropriate.

## Grading

Modules will be graded S/U. As part of the cumulative modular program (Cump2) this module will contribute to that numeric score. These criteria are given for reference.

Programming style- 10%

Simple first in first out structure – 30%
- Create all four (for each direction)
- Correctly implement add()
- Correctly implement remove()

Modify the arrival function to put "cars" in the queue using the timestamp – 25%

Modify the departure function to take "cars" from the queue – 25%

Correctly calculate the average wait time for each direction of travel – 20%