# Program #1 Review

***Written by Alexander DuPree***

## Data Structure Performance

Program #1 required the use of an array of linear linked lists to represent vehicles travelling along a one way street. Accompanying the array of linear linked lists was a data structure of our choice to represent the path of the vehicle with pointers to other vehicles in the system. Because of the O(n) time complexity for accessing objects located in the array of lineaer linked lists I chose to implement a dynamic 2D array of Position pointers to allow for quicker access. Each index in the 2D matrix represented a position along the street where null is empty and any pointers pointed to an object in a linear linked lists. This helped achieve an O(1) time complexity for accessing objects in the system.

This dual data structure requirement, however, made certain aspects of the program difficult to design and implement. For example, changing lanes required that the player vehicle checks that the adjacent location is not inhabited by a vehicle already, then the object would have to migrate from it's current linked list to the correct linked list, representing the lane change. Concurrently, the 2D matrix has to update all the new vehicle positions in the matrix. This essentially created a lengthy operation of removal, insertion, and error handling just to support the player switching lanes during the simulation. I believe that this could have been simplified if only one data structure was used for this program instead of two. If the 2D array was the only data structure used then it wouldn't be necessary to execute the insertion, removal and error handling operations inherent with the linked lists. A lane change could be as simple moving the Position pointer to the correct index.

## Object Oriented Design

The design of program #1 emphasized the single responsibility principle for each class utilized during the program. As such each class had to have a clear responsibility and implement methods only related to that responsibility; and for the most part I feel like this was accomplished in my design. My design of program #1 consisted mostly of two parts: a traffic manager containing the array of linear linked lists and the Position abstract base class along with its derived classes.

The Position abstract base class served as the common interface for objects in our simulation. The vehicle, player vehicle, and traffic signal classes are all derived from the Position base class. This is because any object in our system had to have positional data accessible so that it could be placed in the appropiate index in the 2D matrix. However, each object updated it's own position differently. For example, the traffic signal class did not update it's position at all, as street lights should not move. However, the vehicle and player vehicle class both needed to move through our simulation in different ways. This created a perfect opportunity for a Position abstract base class that defined pure virtual methods for displaying and updating positional data.

Bringing the program together was the traffic manager class. The traffic manager's responsibility was to manage the creation, destruction, and lane information of all Position objects in the system. The traffic manager utilized dynamic binding and polymorphism to hold all derived objects in our system together as Position pointers. Both the array of linear linked lists and the 2D matrix stored Position pointers and had no knowledge of the underlying object's type. Composed into the traffic manager was the traffic matrix class. The traffic matrix primary responsibility was to trigger position updates on all objects and print to the console the graphical representation of the street. This is where the single responsibility principle was slightly blurred. To assist in the traffic managers job, and vice versa, the traffic matrix had to signal up to the manager that a lane change had occured. Because each linked list in the array represents a specific lane, when a lane change occured the manager had to find, remove, and insert the offending position object into the correct linked list.

## GNU Debugger Discussion

The GDB was an invaluable tool during the testing and production of program #1's code. I also utilized the catch2 unit testing framework to establish a suite of tests for my program. This test suite coupled with the GDB allowed me to catch and pinpoint errors in my code almost immediately. If a certain test failed after a lengthy refactor I could fire up GDB, set a break point at the test and step through each line of could to identify the error. The debugger, coupled with unit tests, was able to exponentially increase my productivity as a programmer as I spent more time writing code and less time debugging.