Module E

Goal-

Use pointers to build a linked structure.

We are beginning a new cumulative module that will simulate a simple traffic intersection.

We will start by building the logic that controls the color each light displays. We will build a state transition diagram (STD) for the intersection (see the diagram at the end of this document). "State" is the status of all data in a program or system. In this case we have four lights: N, S, E, & W. They have 3 colors: R, G, & Y. For our simple intersection they are paired: E with W and N with S. If E-W is G or Y then NS must be R and vice versa. So what is a state? Here is an example of the state for EW-G/NS-R:

EW	G
NS	R
nextstate	Pointer to next node
Data and Activity Related to this State:	
NrOfVehiclesAtELight	NrOfVehiclesAtWLight
nrOfVehiclesArrive	nrOfVehiclesdepart

You will have 4 states. What are they? To simplify things use a single class for each of them. **Define 1 State class and create 4 objects**. You will need direction1 and direction2 rather than explicit E, W, N, and S. You ill need data members and accessors to record and report which directions are managed by that state; direction1 = E and so on. You need data members and accessors to record and report the number of vehicles waiting in each lane.

NOTE: Depending upon the context I use node, state, and class interchangeably. You have a class that is a node in a linked structure that represents a state in a STD. I suggest you call your class State.

You will have a pointer currentState that indicates which state the intersection is in. When time expires it follows the link to the next state (node). When the simulation first enters a node it will call nrOfVehiclesArrive() and nrOfVehiclesdepart(). for each direction that is G. The difference (arrive – depart) is the number of vehicles that added to or subtracted from the appropriate NrOfVehiclesAtLight counter. **You will only have 5 pointers in your program**: the next pointer in each Node class (4), and the current pointer to the active node/state.

Randomness. For this module nrOfVehiclesdepart() will return a random number with a small uniform distribution, e.g. 4, 5, 6, 7, or 8. Sadly some people never see the light has changed. ☹ For this module nrOfVehiclesArrive() should return a wider range of values that probably should not have a uniform distribution. UH? WatHeSay? A uniform distribution of 4, 5, 6, 7, or 8 means each number has the same chance of occurring. A random draw for each is equally likely. Think of a 5-sided die ranging from 4 to 8. nrOfVehiclesArrive() would be normally distributed, i.e. the middle value is more likely. You can think of rolling two 6-sided dice. The results range from 2 to 12. 2 and 12 each have a 1/36 chance of occurring while a 7 has a 1/6 chance. Feel free to adjust these ranges to produce interesting results. However, **arrival and departure must be random**!

Time. This is a very simple intersection. All of the lights are on for the same length of time. Actually, the yellow lights do nothing. You will have a loop that goes through the states. **Each iteration of the loop is one time unit**. When to stop the simulation? At the beginning, prompt the user and make some suggestions. If the intersection cycles once per minute that's only 600 iterations for a 10 hour day. We do not vary the arrival and departure rates (to simulate rush hour or non-rush times) so setting it to 10,000 might be fun but not realistic.

Data collection. This part is up to you to design. **You will calculate the average wait time for each traffic lane**. First, decide what that means. ☺ Then determine what data to collect. At the end of the simulation report average wait time per lane for the total simulation period. You will collect all data in the appropriate node (i.e. class). You will use accessors in your program to read the data to display.

So you and the grader can determine the simulation is working correctly **display the total number of cars arriving and departing in each lane each cycle of the light**. This information is not used elsewhere in the program so it can simply be printed to the screen So you and the grader can read the output you will **put in a slight delay**. When debugging and testing you might need to see the node-by-node operation of the simulation.

DESIGN using decomposition and incremental development- There are 2 parts to the program: the state transition diagram, and the activity in the state. This simulation uses a simple STD; there are 4 states and no branching. Each state is a Node class. You need a loop that just follows the pointer to the next node. To design the actions in each state sit down and work through how you handle traffic arriving, traffic getting through the intersection, keeping track of the cars still in each lane and what function(s) you need deliver data back to your main program. **All data and activity related to the traffic and traffic lanes MUST be done in the State class.**

HINT: See the diagram at the end fo this document.

NOTE: We will assume that all vehicles, bicycles, pedestrians, skateboarders, scooters, horse and buggies, skiers, and troikas obey the traffic laws. Specifically, cars stop on Yellow. ☺

NOTE: This may seem complicated for a simple intersection and it is. It lays the foundation for future changes and enhancements. You will be making at least one major change to help demonstrate (functional or object) decomposition in your design.

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%

Transitions in the STD work correctly- 20%

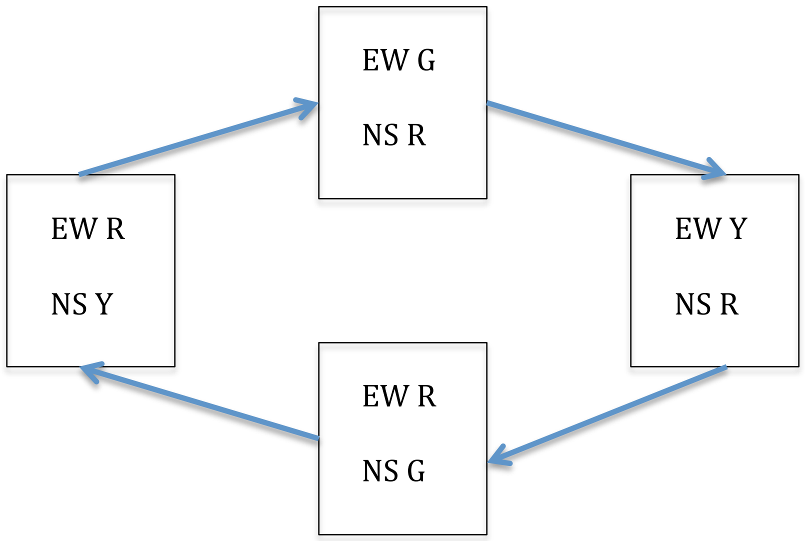
Traffic arrives in each lane correctly- 20%

Traffic departs each lane correctly- 20%

Data collected and displayed demonstrates correct operation of the intersection - 20%
includes the cycle by cycle display as well as the final average wait time

Submit all files in zip archive- 10%

4-Way Traffic Light STD



You will create an object to hold each state.
In the object you add member data and functions so your program will do something useful.