Overview: In this project, you will work as a team to write a C++ implementation of a traffic intersection simulation. You will work in groups of two. Note that there will be absolutely no extensions to the deadline for the final project!

Details:

- At the start of the simulation, there should be no vehicles on the road.
- Each lane — eastbound (EB), westbound (WB), northbound (NB), southbound (SB) — in the roadway must be made up of individual "sections" appropriately linked together. (Note that, at the intersection, different lanes will have to share the same "section".)
- You must implement three different types of vehicles: cars, SUVs, and delivery trucks.
  - Cars will occupy exactly two sections of the roadway. SUVs will occupy exactly three sections of the roadway. Delivery trucks will occupy exactly four sections of the roadway.
  - A given vehicle may either proceed straight forward through the intersection or turn right. (Challenge: implement left turns for vehicles. This is not required *and should not be attempted until you have completed a full working submission ready version of the project that does not include left turns!)*
  - When a vehicle arrives to a lane (see more below), the vehicle will intend to turn right with probability $0 \leq p_r \leq 1$, turn left with probability $0 \leq p_l \leq 1$, or go straight forward with probability $p_f = 1 - p_r - p_l$. (Note that $p_r + p_l + p_f = 1$.)
  - When a vehicle approaches the intersection and is clear to proceed (see traffic light and section-ahead rules below), it may advance straight or turn right unimpeded. If a vehicle is turning left, it may proceed only if no oncoming vehicle will enter the lane to be crossed until after the turning vehicle has completely exited the intersection. (How do you handle simultaneous left turns?)
- The intersection will be controlled by a traffic light in each of the four directions.
  - The traffic light will be green for a period of time (g clock ticks, when traffic may proceed uninterrupted, subject to left-hand turns), yellow for a period of time (y clock ticks, when traffic may proceed only when the corresponding vehicle can completely exit the intersection by the time the light becomes red), and red (r clock ticks) for a period of time.
  - The length of the red for the EB/WB lights will be the sum of the lengths of green and yellow for the NB/SB lights, and vice-versa.
  - The NB/SB traffic lights will be exactly synchronized with one another. That is, the length of green, yellow, and red (since the latter depends on the EB/WB green + yellow lengths) will be identical for the NB and SB lanes. Similarly, the EB and WB lights will be exactly synchronized.
  - The NB/SB lights need not have the same green/yellow lengths as the EB/WB lights.
  - When the NB/SB lights are green or yellow, the EB/WB lights must be red, and vice-versa.
- Your simulation will be driven by an integer-valued clock.
  - At each clock tick, your simulation must attempt to advance any vehicles present exactly one "section" (either straight, right, or left). You must do this for all four directions of travel.
  - Your simulation must only proceed for one single tick at a time. Just as with `testAnimator`, the user must press a key or enter before the simulation moves to the next tick. You may think this is minor, but it is not — when I grade your project, I want to be able to move slowly from tick to tick.
  - A vehicle may proceed only if the road section ahead in its intended direction of travel is open, and subject to rules of the traffic light and (if appropriate) the rules of turning.

- At each clock tick, you must advance the traffic lights appropriately toward an eventual change in color.
- At each clock tick, a new vehicle will arrive at the NB lane with probability p_{nb}, where $0 \leq p_{nb} \leq 1$. Similarly for the other three lanes, $0 \leq p_{sb} \leq 1$, $0 \leq p_{eb} \leq 1$, and $0 \leq p_{wb} \leq 1$. Stated differently by example, if $p_{nb} = p_{eb} = 1$ and $p_{sb} = p_{wb} = 0$, then at each clock tick, a new vehicle will arrive at the NB and EB lanes, but no cars will ever arrive at the SB and WB lanes.
- When a vehicle arrives, the probability of that vehicle being a car will be p_c, of being an SUV will be p_s, and of being a delivery truck will be $p_t = 1 - p_c - p_s$. (Note that $p_c + p_s + p_t = 1$.)
- When a vehicle arrives to a particular lane, if there is an open section at the entrance to that lane, the vehicle will begin to "enter the lane". (One possible approach: if the vehicle is a car, 1/2 of the car will be in that section; if the vehicle is a delivery truck, 1/4 of the truck will be in that section. The remainder of the vehicle will be "in the system" but not on any of your implemented road sections, i.e., hanging off the entry end of your simulated lane of traffic.)

Random Number Generation:

- For implementing the probabilities discussed above, you must use a single instance of the `mt19937` class provided in the C++ `random` library. Because you will need to generate floating-point numbers at random between 0 and 1, you will need to use the templated `uniform_real_distribution` class also available in `random`. Some details will be discussed in class, but below is an *example* of generating ten random numbers between 0 and 1 within a simple `main` function. This example code is not intended to be part of your project, but rather to show you how to generate random numbers in C++ (something that you *will* need for the project).

```
#include <iostream>
#include <random>

int main()
{
    int initialSeed = 8675309;

    std::mt19937 rng;       // construct an instance of mt19937
    std::uniform_real_distribution<double> rand_double(0.0, 1.0);
        // rand_double will use rng to generate uniform(0,1) variates

    rng.seed(initialSeed);  // set the initial seed of the rng

    for (int i = 0; i < 10; i++)
    {
        std::cout << rand_double(rng) << std::endl;
    }
    return 0;
}
```

- You should not have multiple instances of `mt19937` in your implementation. You may want to think about having a separate class to handle the random number generation you need.
- Keep in mind that whenever you set the seed to its initial value, your generator will then start producing exactly the same sequence of numbers that it did initially — so set the seed only once, at the start of your simulation. For example, the two loops in the code excerpt below will produce exactly the same three random numbers.

```
std::mt19937 rng;       // construct an instance of mt19937
std::uniform_real_distribution<double> rand_double(0.0, 1.0);

rng.seed(initialSeed);  // set the initial seed of the rng
for (int i = 0; i < 3; i++) {
    std::cout << rand_double(rng) << std::endl;
}

rng.seed(initialSeed);  // set the initial seed of the rng
for (int i = 0; i < 3; i++) {
    std::cout << rand_double(rng) << std::endl;
}
```

Requirements for Execution:

- Your code *must* build and run successfully on the department Linux cluster. This is where I will build and run it during grading.
- You must provide a Makefile that I can use to compile your code.
- You must read appropriate input parameter values for your simulation from an input file (required format of the file will be discussed in lab, and an example is provided). Note that your simulation must correctly interpret and react appropriately to the configuration parameters in the input file! (You may also assume that these configuration parameters will be reasonable. I will not, for example, try to set the roadway so that there is only one section, excluding the intersection, in each lane. Note that I *do* consider it reasonable to set a turn probability or vehicle type probability to 0 or 1.)
- You must accept command-line arguments similar to the following:

  ```
  % ./simulation [input file] [initial seed]
  ```

  where the first argument is the name of the parameter-values input file and the second is the initial seed passed to your random number generator.
- I will test your code by passing in various input files that I create by hand, testing different scenarios.

Requirements for Code: Part of the goal of this project is to give you some experience using some of the C++ coding techniques we have learned this semester. In particular, your code must meet all of the following specifications

- For every nontrivial class in your code, you must implement a copy constructor, a copy assignment operator, a move constructor, and a move assignment operator, and a destructor. Any non-trivial class must contain all of these. (If you try to avoid this by only using structs, you will not like the resulting grade.)
- Though I am not requiring you to submit formal unit tests, you should **thoroughly** test your code. Your `README` file should list the errors about which you are aware. Errors I find that are not contained in this list will indicate that your testing was not as detailed as it should be.
- Though you certainly may have debugging code in your project, when I test your code, there should be no debugging information or messages output!

Submitting (group):

- Create a tarball containing your Makefile, all classes (including test code), and a README explaining your design decisions, and how to compile and run your code.
- Name the tarball similar to `cmsc240_final_st1ab_st2bc.tgz` which includes all students' netids in the name of the tarball.
- Submit your tarball using the usual method and email address

  `final_p.9unn9fp11a1ghaym@u.box.com`

Submitting (individual): Each group member must also complete an anonymous evaluation of all group members.

- Create a plaintext file named similar to `cmsc240_final_eval_st1ab.txt`, replacing `st1ab` with your netid.
- In the file, include the following:
  - Describe in outline form what you did on the project. Include interactions with others or shared activities. Include any comments concerning what you might have done differently in retrospect.
  - For each member of the team other than yourself:
    (a) Provide the team member's name;
    (b) Give one of the ratings listed below based on your direct, personal knowledge (if you do not feel you know enough about what the team member did to make a rating, use the "no evaluation" rating); and
    (c) Provide comments to justify the rating.

    Ratings:

    "Did fair share of work"
    "Did more than fair share of work"
    "Did less than fair share of work"
    "No evaluation"
- Upload the plaintext (not Word) file directly into the personal Box folder that you share with me.

All Materials Due: 5:00 pm on Thursday, April 20, 2023