CS115 - Computer Simulation, Assignment #1 – Train Unloading Dock
Due Last Lecture of Week 3 at START of class
Note you CANNOT use CSIM – must use a non-simulation language

In this assignment, you will write a simulation of a train unloading dock. Trains arrive at the station as a Poisson process on average once every 10 hours. Each train takes between 3.5 and 4.5 hours, uniformly at random, to unload. If the loading dock is busy, then trains wait in a first-come, first-served queue outside the loading dock for the currently unloading train to finish. Negligible time passes between the departure of one train, and the entry of the next train (if any) into the loading dock---unless the entering train has no crew (see below).

There are a number of complications. Each train has a crew that, by union regulations, cannot work more than 12 hours at a time. When a train arrives at the station, the crew's remaining work time is uniformly distributed at random between 6 and 11 hours. When a crew abandons their train at the end of their shift, they are said to have "hogged out". A train whose crew has hogged out cannot be moved, and so if a hogged-out train is at the front of the queue and the train in front finishes unloading, it cannot be moved into the loading dock until a replacement crew arrives (crews from other trains cannot be used). Furthermore, a train that is already in the loading dock cannot be unloaded in the absence of its crew, so once the crew hogs out, unloading must stop temporarily until the next crew arrives for that train. This means that the unloading dock can be idle even if there is a train in it, and even if it is empty and a (hogged-out) train is at the front of the queue.

Once a train's crew has hogged out, the arrival of a replacement crew takes between 2.5 and 3.5 hours, uniformly at random. However, due to union regulations, the new crew's 12-hour clock starts ticking as soon as they are called in for replacement (i.e., at the instant the previous crew hogged out); i.e., their 2.5-3.5 hour travel time counts as part of their 12-hour shift.

You will simulate for 7200 hours, and output the following statistics at the end:

1. Total number of trains served.
2. Average and maximum over trains of the time-in-system.
3. The percentage of time the loading dock spent busy, idle, and hogged-out (does this add to 100%? Why or why not?)
4. Time average and maximum number of trains in the queue.
5. A histogram of the number of trains that hogged out 0, 1, 2, etc times.

Bonus work for extra credit: If all other parameters are fixed, estimate the long-term maximum train arrival rate that this system can accommodate before overloading. Perform the estimate in two ways: first, a *completely* paper-and-pencil analysis; second, using your simulation. Your grade on this part will depend partly on how good your two estimates agree. Explain your rationale in both cases; especially, in the case of the simulation, explain what criteria you used to decide if the system was "overloaded".

Input Specification: We *will* run your code, but to make it easy for the grader, we need everybody to adhere to the following guidelines: your program should take two command-line arguments, in the following order: average train inter-arrival time, total simulation time. Thus, for example, if your program is written in C and compiled to an executable called "train", then to run it with the default parameters above, I should be able to run it on my Unix command line as

```
% ./train 10.0 7200.0
```

Output Specification: Your program should print one line for every event that gets called. I want to be able to follow what's happening in your code. Each train and each crew should be assigned an increasing integer ID. Output lines should resemble

Time 0.00: train 0 arrival for 4.45h of unloading; crew 0 with 7.80h before hogout, Q=0
Time 0.00: train 0 entering dock for 3.72h of unloading, crew 0 with 7.80h before hogout
Time 0.56: train 1 arrival for 4.33h of unloading; crew 1 with 6.12h before hogout, Q=1
Time 3.72: train 0 departing, Q=1
Time 3.72: train 1 entering dock for 4.33h of unloading, crew 1 with 2.96h before hogout
Time 6.68: train 1: crew 1 hogged out during service; SERVER HOGGED
Time 9.43: train 1: replacement crew 2 arrives, SERVER UNHOGGED

Submission: Submit a brief write-up of your results, along with your source code and its output for the default parameters. Extra credit students should submit their analysis and some input/output pairs that justify their simulation-based conclusions. Submit both a paper printout to me in class, and also electronically to http://checkmate.ics.uci.edu.

Grading: You can use any non-simulation language (ie., any language not already designed for simulation), but it must allow command-line execution similar to the above, and I must be able to run it on my Linux box. This probably eliminates most proprietary languages, such as Matlab. However, if you want to use anything "weird" (i.e., anything other than Pascal, Fortran, C, C++, Java, Lisp, or Scheme), please clear it with me first. In the worst case, I may ask you to run it for me, in my office, in front of my eyes, if I can't figure out how to run your language myself.

Your code should be "pretty", which means easily understood and maintainable by another programmer. This is of course subjective, but it should be well indented, and well commented inside, such that, if we were fellow employees and I had to change your code, I wouldn't be cursing your birth after several hours (or days) of trying to figure it out. It should be easy-to-read; the variables should have meaningful (but not overly verbose) names; the comments should clarify tricky points but not obvious ones. (I've seen the comment "add one to x" beside "x++"; that qualifies as an unnecessary comment. A better comment would tell us WHY x is being incremented, if it's not obvious.) The prettiness (i.e., understandability, readability, and maintainability) of your code, by the grader's judgment alone, will count as 25% of your grade.

Your code should be correct. We will judge the correctness of your code both by reading it, and based on tracing its output events and final statistics. Correctness of the traces and the output will count for 50% of the grade. The write-up will count for the remaining 25%.

The extra work for will count for *another* 10% of their grade (5% each for the paper and simulation analysis), so that their maximum grade will be 110%.