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 72,000 hours, and output the following statistics at the end:

1. Total number of trains served.
2. Average and maximum of the time-in-system over trains.
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.

Input Specification: We *will* run your code, but to make it easy for the grader, we need everybody to adhere to the following guidelines: create a makefile that builds your program, and your program should take either one or two command-line arguments. When given only one argument, your program should read that argument as the *file path* to the *pre-determined train arrival schedule* (described below). When given two arguments, the first argument must be the *average train inter-arrival time*, and the second argument must be the *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:

```
% make
% ./train 10.0 72000
```

or

```
% make
% ./train schedule.txt
```

If you are using a language that does not run like the above (e.g. Python "`python train.py 10.0 72000.0`") create a shell script or program wrapper that takes in the arguments and runs your code as above. Also, if you are using a language that does not require building/compiling (e.g. Python), just create a makefile with no targets:

```
target: ;
```

The pre-determined train arrival schedule contains three space-delimited columns: *arrival times*, *unloading times*, and *remaining crew hours* (in that order), with each arrival event on a new line:

```
0.02013 3.70 8.92
8.12 4.12 10.10
12.52 3.98 7.82
...
1210.0 4.12 9.21
```

The simulation ends after the last train has departed.

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. The final statistics should come after the simulation output and closely match the specified format. Output lines should be identical the following example (except the numbers, of course, which will be different for each run):

```
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)
...
Time 1234.00: simulation ended

Statistics
----------
Total number of trains served: 512
Average time-in-system per train: 3.141593h
Maximum time-in-system per train: 6.283186h
Dock idle percentage: 64.00%
Dock busy percentage: 32.00%
Dock hogged-out percentage: 16.00%
Average time-in-queue over trains: 2.718282h
Maximum number of trains in queue: 2
Average idle time per train: 0.707107h
Histogram of hogout count per train:
[0]: 512
[1]: 128
[2]: 32
```

. . . (show as many bins as necessary)

Numbers can have any number of decimal places (valid examples include 13 and 3.14159), but no scientific notation. A script to automatically check the output will be provided feedback on I/O specification compliance.

Submission: Submit a brief write-up of your results, along with your source code and specific sections of the output (only the first pages and the last page) for the default parameters. There should be no more than 10 pages. Submit both a paper printout to me in class, and also electronically via the submit command on ICS openlab. The submit command can be found in ~cs146/bin/submit. Run it with no arguments for help.

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 an openlab computer. 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 all cases an executable file called "train" must be built by your makefile so we can run it above---**I don't care what language the simulation is written in, but a wrapper called "train" that takes two arguments *must* be present.**

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%. Your write-up should describe your simulation model, how you implemented important bits like the entities, their relationships, and the event scheduling system, as well as comment on whether your output "makes sense". Think about how you'd convince me your program is correct. Do the numbers make sense given the input? (Eg, with 72,000 hours and trains arriving every 10 hours on overage, you'd expect about 7200 trains. Do you see that? What else can you provide as reasonable grounds to assume your simulation is trustworthy?)