The objective of this assignment is to provide some hands-on experience on **multithreaded programming** and **thread synchronization**. You will design and implement a multi-threaded program to control traffic on an imaginary narrow bridge.

Traffic Patterns: The traffic will consist of cars and trucks. A vehicle (car or truck) may be traveling south or north. The direction and the type of each vehicle will be determined according to given probability distributions at the time of its arrival. Cars and trucks will have a weight of 100 and 300 units each, respectively. Each vehicle will take 2 seconds to cross the bridge.

Bridge Restrictions:

R1. Since the bridge has only one lane, the traffic may cross the bridge in one direction at a time.

R2. Due to construction constraints, the total weight of vehicles on the bridge cannot exceed 750 units at any time.

Violating the restriction R1 will cause collisions, while violating R2 will result in the collapse of bridge. **Your program should prevent collisions and the collapse of the bridge**.

Traffic Control Policies: Traffic flow will be controlled according to the following rules.

P1. Once the traffic starts crossing the bridge in one direction, any other vehicles arriving at the bridge in the direction of current traffic will have priority over any vehicle that may be waiting/arriving at the other end of the bridge, **subject to the fairness condition P2 below**.

P2. If there are vehicles waiting at one end of the bridge in one direction, then *at most 6 consecutive vehicles* may cross the bridge from the other end (in other direction). In other words, a vehicle which is (or, arrives) at the "head" of the queue at one side of the bridge should never wait for more than 6 vehicles crossing the bridge in the other direction[1].

P3. Subject to restrictions R1-R2 and policies P1-P2, the bridge capacity must be fully used when there is sufficient traffic. In other words, no vehicle should incur unnecessary delay (for example, having all the vehicles cross the bridge one by one regardless of their types is not acceptable). The *deadlocks* should be also prevented.

[1]Suppose a vehicle V arrives at/reaches the head of the queue in a given direction at time t. At that point, X vehicles may be currently *crossing* the bridge in the opposite direction. These X vehicles should be also counted towards the limit of six vehicles to ensure fairness.

Representing Vehicles as Threads: You will represent each vehicle by one thread, which executes the procedure Vehicle-Routine upon arrival at the bridge:

```
Vehicle-Routine(parameter-list)
{
  Arrive(...)
  Cross(...)
  Leave(...)
}
```

The *parameter-list* in Vehicle-Routine above should contain at least *vehicle-id* (an integer uniquely identifying the vehicle), *vehicle-type* (Car or Truck) and *direction* (southbound or northbound). You are free to use additional parameters and to determine the parameter lists of procedures Arrive, Cross and Leave.

The Arrive procedure must check all the traffic/bridge restrictions and it must not return until it is safe for the vehicle to cross the bridge. The Cross procedure will just delay for 2 seconds for each vehicle. Finally, the Leave procedure must take additional steps to let the traffic control mechanism update its internal state and to let additional vehicles to cross the bridge, if any. **These procedures must print all the information necessary to follow the arrival and departure patterns as well as the list of waiting queues and crossing vehicles.** Your program should print the list of the vehicles that have arrived but not started to cross the bridge (that is, waiting vehicles) in a clear way. It should also print the total weight of vehicles currently crossing the brdige. The format shown in the following example can be adopted:

```
Vehicle #5 (Northbound, Type: Car) arrived.
Vehicle #6 (Northbound, Type: Truck) arrived.
Vehicle #7 (Southbound, Type: Truck) arrived.
.......
Vehicle #5 is now crossing the bridge.
Vehicles on the bridge: [#3 (Northbound, Type: Truck), #4 (Northbound, Type: Car),
#5 (Northbound, Type: Car]. Total Weight: 500
Waiting vehicles (Northbound): [#6 (Type: Truck)]
Waiting vehicles (Southbound): [#7 (Type: Truck) #8 (Type: Car), #9 (Type: Truck)]
.......
Vehicle #5 exited the bridge. Total Weight: .......
.......
```

You should remember that it is your responsibility to display the traffic flow and list of waiting vehicles clearly, unambiguously, and in the proper order.

Programming Language, Thread Libraries: For this assignment, you can use C, C++ or Java as the programming language. You can use any multithreaded programming library; however you are encouraged to use Pthreads (for C/C++) or Java Threads (for Java), since you are likely to find a large number of references for these two options. Pthreads and Java threads are also available at IT&E lab machines. Your project must be developed on (or ported to) a Unix/Linux system. Your program *must* compile and run on either mason.gmu.edu or zeus.ite.gmu.edu – it will be tested and graded on these systems.

Synchronization Primitives: You will write the procedures Arrive, Cross and Leave using *mutex locks* and *condition variables*. You can use the equivalent thread synchronization facilities (wait() and notify()) if you develop your program in Java. Recent Java releases provide support for condition variables and mutex locks in java.util.concurrent package as well.

Your solution must not employ busy waiting. In this assignment, for full credit, you should only use pthread_cond_signal() (case of Pthreads) and (notify() or signal() (case of Java Threads)) for signalling any (blocked) threads (on a condition variable). In other words, *avoid using* pthread_cond_broadcast() function, notifyAll() method, signalAll() method, or any other primitive that releases all the blocked threads at once. Because of the thread scheduling mechanism of the library you are using, you may occasionally observe that the vehicles do not leave the bridge in the same order they entered. You do not have to fix this problem.

Running Your Program with Specific Schedules: In this assignment, you have to run your program for 5 (five) vehicle arrival schedules given below:

1. 20 : DELAY (10) : 10 Car/Truck Probability: [1.0, 0.0]
2. 20: DELAY (10) : 10 Car/Truck Probability: [0.2, 0.8]
3. 30 Car/Truck Probability: [0.8, 0.2]
4. 15 : DELAY(45) : 15 Car/Truck Probability: [0.7, 0.3]
5. 10: DELAY(2) : 10 : DELAY (2): 10 Car/Truck Probability: [0.5, 0.5]

Each schedule will include a number of *groups of vehicles* separated by the DELAY keyword. The integer in the specification of each group represents the number of vehicles arriving *simultaneously* at the bridge in that group. The numbers in parantheses after the DELAY keyword indicate the delay before the next arrival(s). For example, under schedule (1) 20 vehicles arrive at the bridge at the start of the experiment, 10 more vehicles arrive 10 seconds after the arrival of the first 20 vehicles and so on. Under each schedule, 30 vehicles arrive at the bridge during the course of the experiment.

The type and direction of each vehicle will be determined randomly when it arrives, according to the specific probabilities. **For all five schedules above, assume that the probabilities of having a southbound or northbound vehicle are equal (i.e. both are 0.50). The type of each vehicle will be determined according to the Car/Truck probabilities given at the end of each schedule.** For example, the schedule (2) is to be executed with the Car/Truck probability [0.2, 0.8], meaning that an arriving vehicle will be a car with the probability of 20%, while it will be a truck with the probability of 80%. Observe that the schedule (1) represents a car-only traffic pattern. For each arriving vehicle, first determine its direction and then its type according to these probabilities.

You can use your system library functions to generate random numbers. Note that simply multiplying the total number of vehicles by the Car/Truck probabilities will NOT give the number of cars and vehicles in that group: you should generate a random number when determining each vehicle's direction. For example, if there are 10 vehicles in a group and the Car/Truck probability is given by 0.8/0.2, this does not mean that you can immediately set the number of car and vehicles to 8 and 2, respectively. The same applies to its type. The incoming vehicles must be generated one after the other without any delay, unless the schedule calls for an explicit delay through DELAY statement.

Entering Schedules: You should design a simple user-interface to allow the grader to enter any of the above schedules (or, another schedule). For example, your program may first prompt about the number of groups in the schedule. Then it may get information from the user about the number of vehicles in each group, followed by the DELAY before the next one (when applicable). Finally, it may ask for Car/Truck probability and Northbound/Southbound probabilites for that schedule. Alternatively, your program may read the schedule information from a file, according to a format of your preference. The bottom-line is that the grader should not have to modify your source program and that he should be able to easily enter the information about the schedule of his preference. **Your program should allow the user to enter a schedule other than the schedules shown above, and with new Car/Truck and Northbound/Southbound probabilities.**

Submission: Submissions will have both soft and hard copy components. First, you have to send an e-mail to both *aydin@cs.gmu.edu* and CS 571 TA *cliu6@masonlive.gmu.edu* by 4:30 PM on October 6, with the following components in the attachment.

a. The source codes of your programs. If you have multiple source files, please compress them before submission; the name of the compressed file should be identical to your last name (e.g. smith.zip). *Please do not use Windows-based compression programs such as WinZip or WinRar* – use tools such as zip, gzip, tar or compress (on zeus or mason).
b. A write-up describing the data structures you use and giving the pseudo-code for your solution (note that the source code augmented with comments is not pseudo-code)
c. The output of runs for the schedules given above
d. A README file containing clear instructions on:
   d1. how to compile and run your program
   d2. how to enter information about a schedule through your user interface

The subject of your submission e-mail should read Submission: CS 571 Assignment 1. In the body of your e-mail, include your full name, GMU student id, and the name of the GMU server your programs compile and run on (i.e. mason.gmu.edu or zeus.ite.gmu.edu). You should also include a statement summarizing the known problems in your implementation (if any).

You will receive an acknowledgement when your soft copy submission is received by us. You should also submit the hard copy components of the items (a),(b), and (c) above (namely, the source code, the write-up and the sample output files) to the instructor by October 6, 4:30 PM. Failure to follow the submission guidelines may result in penalties.

Resources: You are strongly encouraged to refer to Multithreaded programming links available at `http://www.cs.gmu.edu/~aydin/cs571/resources.html`. Please direct your programming and system questions to CS 571 TA Changwei Liu (*cliu6@masonlive.gmu.edu*), using the keyword "CS 571 Assignment 1" in the subject line. However, before contacting TA, make every effort to check thoroughly the web tutorials and resources provided in the address above. The CS 571 TA prepared also a web page with multi-threaded program examples that compile and run on GMU systems (`http://mason.gmu.edu/~cliu6/cs571/cs571.html`). CS 571 TA holds regular office hours on Monday 5:00-6:30 PM and Wednesday 5:00-6:30 PM in Engineering Building Rm. 4456. Conceptual questions about the project can be directed to the instructor (*aydin@cs.gmu.edu*).

Suggestions: First, you will have to study the basics of multithreaded programming in C or Java (see Resources above). It is strongly suggested that you solve the problem with pen and paper before starting to code. An incremental approach where you first address the simple case of the single-direction traffic may be helpful. Then you can focus on the case of bi-directional traffic. Keep in mind that the multi-threaded programs typically require non-trivial debugging effort as well. **It is important not to postpone the project to the very last days unless you have prior experience in multithreaded programming.**

Grading: The relative weights of grading components will be approximately as follows (the source code and the output of the five schedules are necessary for grading):

Write-up: 20 points
Correct Design and Implementation: 80 points
Total: 100 points

Correct implementation of the bridge restrictions is a necessary condition for traffic control policy implementation. Be advised that simple implementations that effectively ignore traffic control policies P1, P2 or P3 will result in significant grade loss.

Late and Incomplete Submission Penalties: Both the soft and hard copy components should be submitted by the submission deadline to avoid the late penalty. Late penalty for Assignment 1 will be 15% for each day. Submissions that are late by five days or more will not be accepted. Please plan in advance.