ASSIGNMENT 2 CMPT 300 SPRING 2019
PLEASE CHECK ASSIGNMENT WEB PAGE FOR DUE DATES

1. The code below initializes a shared (global) variable tally that will be shared between two threads.

```
int tally=4096;
void Double( ) {
    int count;
    for (count = 0; count < 12; count++) {
            tally *= 2;
    }}

void Half( ) {
    int count;
    for (count = 0; count < 12; count++) {
            tally /= 2;
    }}
```

The main program using the functions illustrated above sets the shared (global) variable tally to 4096, after setting the initial value the main function does not use the global variable tally. Then, the main program starts two threads. One thread runs the function Half(), the second thread runs the function Double(). The functions Half() and Double() are the only functions that access the global variable tally.

Assume the global variable is unprotected. Determine the lower bound and upper bound on the final value of the global variable tally after both threads have run to completion. Assume the two threads can execute at any relative speed. Also assume that a value can only be divided or multiplied by 2 after it has been loaded into a register by a separate machine instruction. (That is tally /= 2 is executed in three machine instructions, load tally into a register, divide by 2, then load the new value of tally back into variable in memory). You may assume that all other instructions (other than changing the value of tally) are never interrupted between the load and increment or between the increment and load back steps.

a) **[10 points]** What is the maximum value of tally? Give an example of an order of execution of Half() and Double() that would produce the maximum value.
b) **[10 points]** What is the **minimum** value of tally? Give an example of an order of execution of Half() and Double() that would produce the minimum value.
c) **[15 points]** What order could the processes run in to produce a value of the global variable tally of 1024 after both processes had completed?

2. [65 points] Consider the following problem; there is a ferry dock on each side of a river. We will call this dock on this side of the river the ferry's home dock. We will call the dock on the other side of the river the ferry's destination dock. The ferry's destination dock is downstream of the ferry's home dock. At the ferry's home dock vehicles containing passengers load on the ferry to travel across the river. The ferry cannot leave the ferry's home dock until a full load of vehicles has been loaded. At the ferry's destination dock the vehicles unload from the ferry. The ferry cannot leave the unloading ferry dock until after all vehicles on the ferry have unloaded. The ferry needs to return to the ferry's home dock empty in order to

make it upstream successfully. We need to consider vehicles traveling in one direction only from the ferry's home dock to the ferry's destination dock.
You will write C code to implement the problem. Your code will be implemented using the libraries (pthread.h, semaphore.h) plus other needed libraries. Do not use any shared memory (shm.h). Make sure all threads are joined before your main program terminates. Remember that sleep() is not thread safe and should not be used in a threaded program. If you must put a thread to sleep use usleep() or nanosleep(), however, it is usually better to use a mutex instead of sleeping.

You will declare counters and semaphores that are accessible by all threads. The following is a minimal list, you may add more if you need to. You will probably need at least one or two more than those in this list.

a) Create counters to count the number of cars and trucks in the waiting queue and the number of cars and trucks loaded, the number of cars and truck saililng and the number of cars and trucks unloaded. (8 counters).
b) Create mutexes to protect each counter (cars queued counter, trucks queued counter, cars loaded counter, trucks loaded counter)
c) Create counting semaphores to represent
    - The cars in the waiting line (cars queued semaphore)
    - The trucks in the waiting line (trucks queued semaphore)
    - The cars loaded onto the ferry (cars loaded semaphore)
    - The trucks loaded onto the ferry (trucks loaded semaphore)
    - The cars sailing (cars sailing semaphore)
    - The trucks sailing (trucks sailing semaphore)
    - The cars unloaded from the ferry (cars unloaded semaphore)
    - The trucks unloaded from the ferry (trucks unloaded semaphore)

<u>You main program will</u>

a) Initialize all needed mutexes and semaphores
b) In the order the quantities are presented below, read user input for
    - o The probability the vehicle will be a truck The probability will be provided as an integer percentage (0<=probability<=100. If the user indicates 33, then there is a 33 percent chance that each vehicle created is a truck and a 67% chance that it is a car.
    - o The maximum value of "the minimum time to wait before creating the next vehicle thread", K. The duration K is taken from a uniform random distribution. K should be specified (in milliseconds) (1000 milliseconds < K < 5000 milliseconds). K is an integer.
    - o The seed for the uniform pseudo random number generator (2 < seed < RAND_MAX, RAND_MAX is provided in <stdlib.h>) seed is an integer.
c) Create a thread to create vehicles. (createVehicle thread)
d) Create a thread to represent the captain of the ferry / the ferry (captain thread)
e) Wait for the threads created above to complete then join them
f) Join the car and truck threads created by the createVehicle thread
g) Destroy all mutexes and semaphores.

<u>Your create vehicle thread will</u>

a) Create car and truck threads. Each car created by this thread will be a separate thread, each truck created by this thread will be a separate thread.
b) The thread should print the following message before creating any vehicles
"CREATEVEHICLE: Vehicle creation thread has been started".
c) Initialize the random number generator (call srand(seed))
d) The initial value of the next arrijjval time to 0

e) For each new vehicle the create vehicle thread will execute the following steps
   1. Calculate the elapsed time.
   2. When the elapsed time exceeds the next arrival time complete steps 3 to 8
   3. Print the message **"CREATEVEHICLE: Elapsed time NNN msec" .** NNN should be replaced by the elapsed time, the clock time since the process started in milliseconds. Use the function provided below to calculate the elapsed time
   4. Determine if the new thread represents a car or a truck by generating an integer drawn from a uniform random distribution between 0 <= random number <= 100. If the number is <=probability the vehicle is a truck then the vehicle is a truck.
   5. Create the car thread or the truck thread
   6. Print one of the messages below
      "CREATEVEHICLE: Created a truck thread"
      "CREATEVEHICLE: Created a car thread"
   7. Determine when the next vehicle should arrive. Draw a random integer from the uniform random distribution to represent the wait till the next vehicle will be created. The wait will be between 1000 msec. <= wait till next vehicle <= K. Remember that the value of K (1000<=K<=5000) was read into the main program. Wait till next vehicle will be the minimum number of milliseconds until the next vehicle is created.
   8. Print the message
      "CREATEVEHICLE: Next arrival time NNN msec"
      NNN should be replaced by the elapsed time at or after which the next vehicle should arrive

<u>Your car threads, each car thread representing one car, will</u>

a) Print the message **"CAR: Car with threadID NNN queued".**
   NNN will be replaced by the value of the threadID for the thread. Use the cars queued semaphores, counters and mutexes as needed to place the car into the queue waiting for the ferry (into the blocked queue of the cars queued semaphore)
b) Print the message **"CAR: Car with threadID NNN leaving the queue to load"**.
c) Tell the captain thread the car is loaded on the ferry (use carsLoaded semaphores, counters and mutexes as needed)
d) Print the message **"CAR: Car with threadID NNN is onboard the ferry"**
   Use the cars sailing semaphores, counters and mutexes as needed to place the car onto the ferry (into the blocked queue of the cars sailing semaphore)
e) Print the message **"CAR: Car with threadID NNN is now unloading"**
f) Tell the captain thread the car has completed unloading (use the cars unloaded semaphores, counters and mutexes as needed)
g) Print the message **"CAR: Car with threadID NNN has unloaded"**
h) Wait until the captain tells the car it can exit the ferry terminal
   Print the message **"CAR: Car with threadID NNN is about to exit"**
i) The car thread has completed and can now terminate. (use pthread_exit())

<u>Your truck threads, each truck thread representing a truck will</u>

Follow the same steps as the threads for cars, using counters and semaphores specifically made for the trucks. Messages printed will be identical to those printed by the cars except that the word car will be replaced with the word truck.

thread representing the captain (or ferry) will

a) Print a message
   "CAPTAIN: Captain thread started"
b) The captain of the ferry is represented by a thread
   - The captain thread is created after the thread to create vehicles.
   - The captain thread begins by waiting until at least 8 vehicles are in the queue. Until that time it will run a loop that tests how many vehicles are waiting once per second.
   - When the captain process leaves the waiting loop the ferry has arrived at the home dock to load the first load of cars and trucks for the day
c) The ferry can take six cars (or equivalent) across the river each trip. One truck takes the same space on the ferry's deck as two cars.
d) On average trucks are heavier than two cars so the Ferry can carry a maximum of 2 trucks.
e) The ferry only makes a trip from the home dock to the destination dock when it is fully loaded (six cars, two trucks and two cars, one truck and four cars).
f) When the captain is ready to load vehicles onto the ferry, the captain thread will first check the number of waiting cars and the number of waiting trucks in the queue. (values of the counters).
g) After the captain has checked the number of cars and trucks, additional cars and trucks may be added to the waiting line. Any vehicles that join the line after the captain checks the numbers of trucks and cars will be considered late arrivals and will be considered only after those that were not late arrivals have been considered.
h) To begin loading at the ferry's home dock the captain will select enough vehicles for a full load.
i) The following rules should be followed when determining which vehicles the captain should select (which vehicles to load):
   1. Select up to two waiting trucks. (the first trucks that arrived)
   2. Select enough waiting cars to fill the ferry. (the first cars that arrived)
   3. If there are not enough waiting cars to fill the ferry check the counters again in case there have been late arrivals.
      - If there is one or more trucks among the late arrivals, and there is room on the ferry for a truck or trucks, then select the late arrival truck or trucks to load
      - If there are no trucks (or the ferry is not full after trucks that fit have been loaded) then select the late arrival cars
   4. Repeat step 3 until the ferry is full
j) When the captain thread selects a vehicle to load it will use the semaphores, counters and mutexes for the cars queued, trucks queued, cars loaded and trucks loaded to remove the car or truck from the blocked queue of the counting semaphore for queued cars or trucks
   - Print one of the following messages
     "CAPTAIN: Car selected for loading"   **"CAPTAIN: Truck selected for loading"**

   - Add the car or truck thread from the blocked queue of the counting semaphore for loaded cars or loaded trucks.
   - Print one of the following messages after the car or truck thread is added to the blocked queue for the cars loaded semaphore
     "CAPTAIN: Captain knows truck is loaded"
     "CAPTAIN: Captain knows car is loaded"
k) The ferry cannot leave until a full load of vehicles has been loaded (both the captain and the vehicles themselves must know that they have been loaded.
   - When all vehicles have loaded and told the captain using the cars loaded or trucks loaded semaphores, mutexes and counters as needed the captain thread can sail
   - Then print the message **"CAPTAIN: Ferry is full, starting to sail"**
l) Sailing to the other dock takes 2 seconds

m) Print the message **“CAPTAIN: Ferry has reached the destination port”**.

n) When the ferry arrives at the unloading ferry dock the captain will
   - Tell the vehicles they can begin to unload using the cars sailing and trucks sailing semaphores, mutexes and counters as needed.
   - Wait for the vehicles to unload using the cars unloaded mutexes, semaphores and counters as needed.
   - Print one of the messages below
     “CAPTAIN: Captain knows a truck has unloaded from the ferry”
     “CAPTAIN: Captain knows a car has unloaded from the ferry”
   - Tell the vehicle it can leave the ferry dock (terminate) and print one of the messages below
     “CAPTAIN: Captain sees a truck leaving the ferry terminal”
     “CAPTAIN: Captain sees a car leaving the ferry terminal”
   - When the captain knows that all vehicles have unloaded the ferry returns to the loading ferry dock for the next load.

o) Your ferry simulation should end when the ferry arrives at the loading ferry dock for the 5th load.