1. Pizza Synchronization

Processes A, B, ... Z (a total of 26 processes) are using the semaphore S in order to enforce mutually exclusive use of the shared variable COUNT. S is initialized to 1 and COUNT is initialized to zero. All processes execute the following code once each.

```c
main() {
    P(S);       /* wait(S) */
    COUNT++;    /* increment the counter */
        /* I can eat a whole pizza */
    V(S);       /* signal(S) */
}
```

Subsequently, Prof. Lo orders the number of pizzas specified in the variable COUNT.

a. Which line or lines of code make up the critical section code? COUNT++
b. What is the maximum number of processes that can be executing in critical section at the same time? **ONE**
c. What is the maximum number of processes that can be blocked on the semaphore S? **25**
d. What is the maximum number of processes that can be in the ready queue at the same time (considering only these 26 processes)? **26 if CPU idle, 25 in one in critical section**
e. If each process executes the above code exactly once, how many pizzas will be ordered? **26**
f. If the P and V calls were accidentally omitted, what is the lowest value that COUNT could receive after all 26 processes executed the COUNT++ instruction exactly once? (How many students will get a pizza in this case?) **ONE**
g. If the P and V calls were accidentally omitted, what is the highest value that COUNT could receive after all 26 processes executed the COUNT++ instruction exactly once? (How much will Professor Lo have spent on pizza assuming each pizza costs $10. Will she be happy about this?) **26, 26*$10, :-<**

2. Write pseudo-code to control the number of students who get to watch the special preview of Avatar 4D version. At most 50 students will be allowed to simultaneously watch this video streamed on the Internet to avoid traffic congestion. Define the semaphore, initialize it, and write the code for each student to execute where the critical section code is "DOWNLOAD Avatar".

```c
semaphore ACOUNT = 50; /* declare semaphore and initialize to 50 */
P(ACOUNT);
Download Avatar-4D
V(ACOUNT);
```
In reality, this situation would not be controlled with semaphores. Why not? Users are downloading across the network; hard to use semaphores across distributed machines that do not share physical memory – long latencies make it very inefficient. Much easier just to control access at the source with a local counter.

3. Write pseudo-code to ensure that processes A and B both complete execution before process C begins. Define the semaphore (s), initialize, and write the code for each student to execute. Note, there is no critical section here. We are using semaphores to ENFORCE PRECEDENCE to coordinate processes. Can this be done with flag variables? Why or why not?

Semaphore WAIT4A = 0, WAIT4B = 0; /* C cannot go until both WAIT4A and WAIT4B gets incremented to +1 */

A: some code … V(WAIT4A); exit(); /* A signals it is done */

B: some code … V(WAIT4B); exit(); /* B signals it is done */

C: P(WAIT4A); P(WAIT4B); some code /* C blocks until both A and B are done */

It is unpredictable what order A,B,C will be scheduled. If C goes first, it will block until WAIT4A and WAIT4B reaches the value +1. This means that both A and B have to signal.

Cannot be done with flag variables unless incrementing and decrementing them are guaranteed to be critical sections.