Name: ______________________________

CSc 422/522 — Examination 1

You may use up to four pages of notes for this exam, but otherwise it is closed book. There are five questions; each is worth 15 points. Graduate and honors students are to answer all questions. Undergraduates are to answer any four—or answer all five and I will count only your four best scores.

You must explain your answer or show how you arrived at it. This is required for full credit and is helpful for partial credit. Do your work on these sheets, using additional sheets if necessary.

(1) Consider the following program:

```
int x = 0;
co ⟨await (x != 0) x = x - 2;⟩
// ⟨await (x != 0) x = x - 3;⟩
// ⟨await (x == 0) x = x + 5;⟩
oc
```

(a) Does the program terminate? If so, what are the possible final values of `x`? If not, why not?

(b) Suppose the `await` statements are changed to `if` statements—namely, that `await` is replaced by `if` and the angle brackets are deleted. Now the program is sure to terminate. What are the possible final values of `x`?

(2) Suppose your machine has the following atomic instruction:

```
flip(lock):  ⟨ lock = (lock + 1) % 2;   # flip the lock
               return (lock); ⟩
```

Someone suggest the following solution to the critical section problem for *two* processes:

```
int lock = 0;

process CS[i = 1 to 2] {
  while (true) {
    while (flip(lock) != 1)
       while (lock != 0) skip;
    critical section;
    lock = 0;
    noncritical section;
  }
}
```

(a) This "solution" does not work. Give an execution order that results in both processes being in their critical sections at the same time.

(b) Suppose that the first line in the `flip` instruction is changed to do addition modulo `3` rather than modulo `2`. Will the solution now work for two processes? Explain.

(3) Given is a matrix of integers `data[n,n]`. Assume that `n` is large, that integer `K` is much less than `n`, and that `K` is a factor of `n`.

Give pseudo-code for a parallel program that uses `K` worker processes to find the `K`'th largest value in matrix `data`. You will probably want to start by figuring out a sequential algorithm, and then parallelize that algorithm. To receive maximum credit, you should have a reasonably efficient algorithm.

At any point that you need a barrier synchronization, just write `barrier` or draw a horizontal line as in class. If you need critical sections, simply put angle brackets around the code that should execute atomically.

4. A *tournament barrier* for eight processes has the following structure:

In the first stage, every process participates (four pairs), in the second stage, four processes participate (two pairs), and in the final stage, two processes participate (one pair). The "winner" of the "tournament" is worker 1 above. When worker 1 finishes the last stage, it lets all the other processes know that every process has arrived at the barrier.

(a) Develop an implementation of a tournament barrier for 8 worker processes. Your code should use flags and/or counters to implement the stages and to signal every process at the end. The barrier should be reusable—namely, the workers should be able to execute the code over and over. Either write a single procedure or sketch the code executed by each worker.

(b) What is the *best case* execution time of your barrier, assuming every worker arrives at the barrier at about the same time. Assume that simple assignment statements take 2 time units and that spin loops or `await` statements take 3 times units.

5. The UNIX kernel provides two *atomic* operations similar to the following:

`sleep():`	block the executing process
`wakeup():`	awaken all blocked processes

A call of `sleep` always blocks the caller. A call of `wakeup` awakens every process that has called `sleep` since the last time that `wakeup` was called.

Develop an implementation of these primitives using semaphores for synchronization. *Be sure to declare and initialize* any variables and semaphores that you use.