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 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 three statements:

S_1: `x := x+y;`

S_2: `y := x-y;`

S_3: `x := x-y;`

Assume that `x` is initially `2` and that `y` is initially `5`. For each of the following, what are the possible final values of `x` and `y`?

(a) S_1; S_2; S_3;

(b) co $\langle S_1 \rangle$ // $\langle S_2 \rangle$ // $\langle S_3 \rangle$ oc

(c) co S_1 // S_2 // S_3 oc

(d) co $\langle$await (x > y) S_1; $S_2\rangle$ // $\langle S_3 \rangle$ oc

(2) It is possible to solve the critical section problem without using special instructions by employing a coordinator process. Each "user" process interacts with the coordinator to indicate when it wants to enter the critical section and when it has exited the critical section. The coordinator "listens" to the user processes and grants permission.

(a) Assume there are `n` user processes, numbered from 1 to `n`. Develop code that the user processes execute for CSenter and CSexit. Also develop code that the coordinator process executes. Use flags and `await` statements for synchronization. *Your solution must be fair in order to receive full credit.*

(b) Explain why your solution ensures mutual exclusion.

(c) Explain why your solution is fair (or why it is not).

(3) Assume there are `n` worker processes, numbered from 1 to `n`. Also assume that our machine has an atomic increment instruction. Consider the following code for a *reusable* `n`-process barrier:

```
int count = 0, go = 0;
```

code exectued by `Worker[1]`:

```
  ⟨await count = n-1⟩;
  count = 0;
  go = 1;
```

code executed by `Worker[2:n]`:

```
  ⟨count++⟩
  ⟨await go = 1⟩
```

(a) What is wrong with the above code? Explain.

(b) Fix the code so that it works. Do not use any more shared variables, but you may introduce local variables. [For partial credit, fix the problem using more shared variables.]

(c) Suppose my code above were correct. Assume all processes arrive at the barrier at the same time. How long does it take before every process can leave the barrier? Count each assignment statement as 1 time unit and each `await` statement as 1 time unit once the condition becomes true.

(4) Suppose there is a circular train track containing T sections, numbered from 1 to T. There are n trains, $n<T$. Each train repeatedly goes around the track. Initially the trains occupy different sections of track. Two trains may not occupy the same section of track at the same time.

(a) Develop code to simulate the actions of each train. (In particular, let each train be a process that goes around the track.) Use semaphores for synchronization. Be sure to declare and initialize every variable that you need. Distinguish between shared and private variables.

(b) Characterize the best case scenario for the movement of the trains, namely what would minimize delays due to synchronization.

(c) Characterize the worst case scenario for the movement of the trains, namely what would maximize delays due to synchronization.

(5) Consider a procedure `exchange(var value: int)`. This procedure is called by any two processes to exchange the values of their arguments. The first process to call `exchange` has to delay; when a second process calls `exchange`, they can exchange their values and go on. The same scenario is repeated by the third and fourth processes to call `exchange` and so on.

Develop code to implement the body of `exchange`. Use semaphores for synchronization. Be sure to declare and initialize every variable that you need. Distinguish between shared and private variables. Explain briefly how your solution works.