Name: ______________________________

CSc 422, Spring 2005 — Examination 1

You may use up to *two* pages of notes for this exam, but otherwise it is closed book. Please put your name on the top of your notes and turn them in with your examination. Do your work on these sheets, using additional sheets if necessary.

The exam is worth 60 points. There are 5 problems; each is worth 12 points. *You must show your work and/or explain your answers.* This is required for full credit and is helpful for partial credit.

1. Consider the following program:

```
int x = 0;
co S1: ⟨await (x != 0) x = x - 2; ⟩
// S2: ⟨await (x == 0) x = x + 3; ⟩
// S3: ⟨await (x == 0) x = x + 2; ⟩
oc
```

(a) This program might terminate, or it might end in deadlock. What execution order leads to termination, and what is the final value of `x` in that state?

(b) What execution order leads to deadlock, and what is the final value of `x` in that state?

(c) Suppose the `await` statements are all replaced by if statements—namely, the word `await` is replaced by `if`, and the angle brackets are deleted. The program is now sure to terminate. What are the possible final values of `x` in this case?

2. Some computers have an instruction that atomically exchanges the values of two memory locations. It is defined as follows:

```
Exchange(int var1, int var2):
   ⟨ int temp; temp = var1; var1 = var2; var2 = temp; ⟩
```

Above, `temp` is an internal machine register.

Using `Exchange`, develop a solution to the critical section problem. In particular, give code for CSenter and CSexit protocols that use the shared `lock` variable declared below. Your solution does not have to be fair.

shared variable: `int lock = 0;`

CSenter:

CSexit:

3. Given is a matrix of integers `data[n][n]`. Assume that the matrix has already been initialized and is shared.

Give high-level pseudo-code for a parallel program that uses `K` worker processes to find the `K` largest values in the matrix. Assume that `K` is much smaller than `n` and that `n` is a multiple of `K`. The result should be stored in a shared array.

All work should be done by the `K` worker processes. Do not use any code (such as `final` in MPD) that executes after the workers terminate. You may use additional shared storage.

Your answer should fit in the space below. If you need barrier synchronization at any point, just draw a line or write "barrier".

4. *The Scary Roller Coaster.* Suppose an amusement park has a roller coaster with two trains that share a circular track. Naturally, the park owners do not want the trains to run into each other. They have asked you to help them synchronize the trains.

Assume the circular track has `T` sections (`T>2`). Train 1 starts on track section 1, and train 2 starts on track section `T`/2. The trains go around the track in the same direction. The speed of each train varies as it goes around, being slower on uphill sections of track and faster on downhill sections.

Model each train as a process, and synchronize their actions so that they do not ever occupy the same section of track at the same time. Use semaphores for synchronization. Be sure to declare the semaphores and show their initial values

5. *The Hungry Birds.* Given are `n` baby birds and one parent bird. The baby birds eat out of a common dish that initially contains `W` worms. Each baby takes one worm out of the dish and eats it. At most one baby is allowed to eat at a time. After eating, a baby bird leaves the dish and sleeps for a while, and then it comes back to eat.

When the dish becomes empty, the baby bird who empties the dish awakens the parent bird, who flies away and gathers `W` more worms, puts them in the dish, and then waits for the dish to become empty again. This pattern repeats forever.

Represent the baby birds and the parent bird as processes, and develop code that simulates their actions. Use semaphores for synchronization. Be sure to declare and initialize all shared variables and semaphores.