Name: ______________________________

CSc 422/522 — Examination 1

You may use up to four pages of notes for this exam, but otherwise it is closed book. Each question is worth 15 points. Graduate and honors students are to answer all five questions. Undergraduates are to answer any four—or answer all five and I will count 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) Recall the Fetch-and-Add instruction, which is defined as follows:

```
FA(var, incr) :
   ⟨ int tmp = var; var = var + incr; return(tmp); ⟩
```

The value of `incr` can be negative.

Using the `FA` instruction, develop an implementation of the `P` and `V` operations on a general semaphore. You will need to use busy waiting in your implementation of `P(s)`.

(2) Consider the following MPD program:

```
resource main()

  process P1 {
    write("line 1"); write("line 2");
  }

  process P2 {
    write("line 3"); write("line 4");
  }

  process P3 {
    write("line 5"); write("line 6");
  }
end
```

(a) How many different outputs could this program produce? Explain your reasoning.

(b) Add semaphores to the program so that the six lines of output are printed in the order 1, 2, 3, 4, 5, 6. Declare and initialize any semaphores you need and add `P` and `V` operations to the above processes.

(3) Consider a Unix `grep` command that searches several files, as in

```
grep pattern f1 f2 ... fn
```

This command prints one line of output for every line in the files that contains `pattern`.

Develop pseudo-code for a parallel program that implements this command using the *bag-of-tasks* programming style. Use an array of `PR` processes; assume that `PR` is less than `n`. State what the tasks are, give a representation for the bag, and declare and initialize shared variables, including synchronization variables. Then give an outline of the actions of the processes. Show how processes use the bag, and ensure that your program terminates cleanly (not in deadlock). It is OK for output lines from different files to be interleaved.

I want to see concurrent programming details, but it is OK to describe sequential parts very briefly.

4. *The Bear and the Honeybees.* Given are `n` honeybees and a hungry bear. They share a pot of honey. The pot is initially empty; its capacity is `H` portions of honey. The bear sleeps until the pot is full, then eats all the honey and goes back to sleep. Each bee repeatedly gathers one portion of honey and puts it in the pot; the bee who fills the pot awakens the bear.

Represent the bear and honeybees as processes and develop code that simulates their actions. Use semaphores for synchronization. Show the declarations and initial values of all semaphores that you use.

5. *Memory Allocation.* A memory allocator has two operations: `request(amount)` and `release(amount)`, where `amount` is a positive integer. When a process calls `request`, it delays until at least `amount` free pages of memory are available. A process returns `amount` pages to the free pool by calling `release`. Pages may be released in different quantities than they are acquired. (We are ignoring the actual addresses of memory pages.)

Develop a monitor that implements `request` and `release`. Initially there are `F` free pages. Use the Signal and Continue discipline. Do not worry about the order in which requests are serviced and do not worry about fairness. However, a request should never be delayed if there are enough free pages.