Name: ______________________________

CSc 422/522 — Examination 2

Spring 2001

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. *For full credit, you must explain your answer or show how you arrived at it.*

1. Develop an implementation of a *reusable* counter barrier for `n` processes using semaphores. In particular, use *only* the following shared variables:

```
int count = 0;             # counter
sem arrive = 1, go = 0;    # semaphores
```

2. Consider the following program in which processes communicate using asynchronous message passing:

```
chan toA(int), toB(int), toC(int);
process A {
  int v1 = 1, new;
  send chC(v1); v1 = v1+1;
  send chB(v1); v1 = v1+1;
  receive chA(new); v1 = max(v1, new+1); v1 = v1+1;
  send chC(v1); v1 = v1+1;
  receive chA(new); v1 = max(v1, new+1); v1 = v1+1;
}
process B {
  int v2 = 1, new;
  send chC(v2); v2 = v2+1;
  receive chB(new); v2 = max(v2, new+1); v2 = v2+1;
  receive chB(new); v2 = max(v2, new+1); v2 = v2+1;
  send chA(v2); v2 = v2+1;
}
process C {
  int v3 = 1, new;
  receive chC(new); v3 = max(v3, new+1); v3 = v3+1;
  receive chC(new); v3 = max(v3, new+1); v3 = v3+1;
  send chA(v3); v3 = v3+1;
  send chB(v3); v3 = v3+1;
  receive chC(new); v3 = max(v3, new+1); v3 = v3+1;
}
```

What are the possible final values of `v1`, `v2`, and `v3`? *Show your work.*

3. Consider a distributed parallel program in which each process has to following outline:

```
declarations of shared channels
process Worker[1:n] {
  local variables
  while (not done) {
    compute
    BARRIER
 }
}
```

(a) Develop code to implement an `n`-process barrier for the `Worker` processes. Assume that the processes communicate using a circular ring; namely, each process communicates only with the neighbors to its right and left. Be sure to declare the channels you use.

(b) Develop a different implementation of the *BARRIER* code, this time using a symmetric solution in which each process communicates with all others, and each takes the same set of actions. Again be sure to declare the channels you need.

4. You are given three processes: `F`, `G`, and `H`. Each has a local array of integers: `f[1:n]`, `g[1:n]`, and `h[1:n]`. All three arrays are sorted in nondecreasing order. At least one value is in all three arrays.

Develop a program in which all three processes interact until each has determined the smallest value in all three arrays. Use message passing for interaction between the processes; do not use shared variables. Messages may contain only one value from `f`, `g`, or `h` at a time—and possibly process ids.

Explain your strategy, and declare the channels you use.

5. A `MatchMaker` process coordinates the rendezvous between pairs of client processes. Assume that there are `n` client processes, with ids from `1` to `n`. The process interact by message passing only—there are no shared variables other than message channels.

For each of the problems below, declare the channels you use, give the client interface to the `MatchMaker`, and give the code for the `Matchmake` process. In both cases, *make the* `MatchMaker` *as simple as possible.*

(a) Assume that each client wishes to rendezvous with any other client. Namely, the `MatchMaker` coordinates a rendezvous between the first two clients who request one, then the next two, and so on.

(b) Now assume that each client specifies the other process with which it wishes to rendezvous. In particular, messages from clients to the `MatchMaker` specify both the sender's id and another process's id. The `MatchMaker` now matches up processes who wish to rendezvous with each other.