Name_____ Student Number_____

CSCI 322, Winter 2013, Final Exam

Question	Points	Score
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	10	
8	10	
9	10	
10	10	
Total	100	

1. Short answer warmup.

   (a) **False sharing** is when (circle one):
   - Two semaphores are used to synchronize the same critical region.
   - Two processes use a single variable.
   - Two processes do not actually share two variables, but the cache hardware treats the two variables as a unit.
   - Two variables are used by more than one process.

   (b) Concurrent programming is (circle one):
   - the same thing as parallel programming.
   - always parallel, but sometimes parallel is not concurrent.
   - sometimes parallel, but sometimes concurrent is not parallel.
   - parallel programming without multiprocessors.

   (c) The circular pipeline implementation of matrix multiplication used (circle one):
   - message passing
   - semaphores
   - monitors
   - critical regions

   (d) Partial correctness means (circle one):
   - the program terminates, but with an incorrect answer.
   - the program computes a correct answer, but does not terminate.
   - the program computes a correct answer, but only if it terminates.
   - the program computes a correct answer and always terminates.

   (e) Total correctness means (circle one):
   - the program terminates, but with an incorrect answer.
   - the program computes a correct answer, but does not terminate.
   - the program computes a correct answer, but only if it terminates.
   - the program computes a correct answer and always terminates.

   Circle the letters to indicate which of the following as a safety **S** or liveness **L** property.

   (f) **S L** mutual exclusion

   (g) **S L** termination

   (h) **S L** absence of deadlock

   (i) **S L** partial correctness

   (j) **S L** all messages reach their destinations

$$\frac{\{P \wedge B\}\ \texttt{S}\ \{Q\},\ (P \wedge \neg B) \Rightarrow Q}{\{P\}\ \texttt{if (B) S;}\ \{Q\}}$$

```
{ m == x }
if (y > m)  m = y;
{(m == x ∧ m >= y) or (m == y ∧ m > x)}
```

The **If Statement Rule** is shown above at left.

2. In the proof outline at right, what booleans correspond to P, B, and Q?

   $P =$

   $B =$

   $Q =$

3. What two propositions would have to be proven to complete the proof outline (proofs not needed)?

   Proposition 1:

   Proposition 2:

4. Give a formal proof of the following. Remember to give the **line numbers that justify each line** whenever a line is inferred from previous lines. Premises for conditional or indirect proofs can just be marked P. Indent subproofs. You do not have to renumber if there are blank lines left in your proof.

Prove: $(P \Rightarrow Q) \Rightarrow ((P \vee R) \Rightarrow (Q \vee R))$

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____
10. _____
11. _____
12. _____
13. _____
14. _____
15. _____
16. _____
17. _____
18. _____
19. _____
20. _____

5. Questions 5, 6, and 7 all refer to the same program, shown at right. Explain how it is possible for this program to terminate with final values of `x` and `y` such that `x != y`. What are the values? You can refer to lines of code with their letters.

```
A   int x = 6, y = 4;
B   co while (x != y) {
C          x = x - 1;
D          y = y + 1;
E      }
F   //
G      ⟨await (x == y);⟩
H      x = 8;
I      y = 4;
J   oc
```

6. Questions 5, 6, and 7 all refer to the same program, shown at right. Explain how it is possible for this program to terminate with final values of `x` and `y` such that `x == y`. What is the value? You can refer to lines of code with their letters.

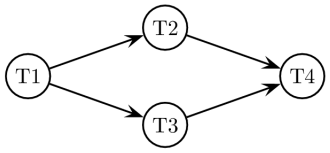
```
A   int x = 6, y = 4;
B   co while (x != y) {
C          x = x - 1;
D          y = y + 1;
E      }
F   //
G      ⟨await (x == y);⟩
H      x = 8;
I      y = 4;
J   oc
```

7. Questions 5, 6, and 7 all refer to the same program, shown at right. Explain how it is possible for this program not to terminate. You can refer to lines of code with their letters.

```
A   int x = 6, y = 4;
B   co while (x != y) {
C          x = x - 1;
D          y = y + 1;
E      }
F   //
G      ⟨await (x == y);⟩
H      x = 8;
I      y = 4;
J   oc
```

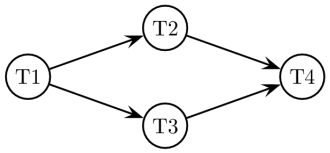
8. Recall that Fetch-and-Add, `FA(var, increment)`, is an atomic function that returns the old value of `var` and adds `increment` to it. Using `FA`, define `P` and `V` operations that will act like semaphores. Assume that memory reads and writes are atomic but that `FA` is the only more powerful atomic operation. You can use busy waiting for delayed processes.

9. Four tasks have precedecessors and successors as in the graph shown. Assume each task executes code similar to the code at right (where i is replaced by 1, 2, 3 or 4). Write code for all four tasks that accomplishes this with semaphores.




```
process Ti:
  while true:
    wait for predecessors;
    print "Ti"
    signal successors;
```

10. Four tasks have precedecessors and successors as in the graph shown. Assume each task executes code similar to the code at right (where i is replaced by 1, 2, 3 or 4). Write code for all four tasks and a single monitor that they share that accomplishes this.




```
process Ti:
  while true:
    wait for predecessors;
    print "Ti"
    signal successors;
```