1. Consider the following concurrent program of two identical processes p and q both incrementing the same shared variable n 10 times. Assume that the underlying hardware provides the normal LOAD and STORE machine instructions so that each individual reference (read or write) to an integer can be considered atomic.

```
int  n = 0;

process p {                          process q {
int temp;                            int temp;
do 10 times {      /*p1              do 10 times {        /*q1
    temp = n;      /*p2                  temp = n;        /*q2
    n = temp+1;    /*p3                  n = temp+1;      /*q3
    }                                    }
}                                     }
```

What are all the possible different final values of n? Construct the scenarios.

2. Consider the following version of the tie-breaker "Peterson's" algorithm for two process critical section problem with atomic await statements of the form: ⟨await A or B⟩.

```
bool inp = false, inq = false;
int last = p;

process p  {                         process q  {
   while (true) {                       while (true) {
p1:    last = p;                     q1:     last = q;
p2:    inp = true;                   q2:     inq = true;
p3:    ⟨await !inq or last==q⟩;      q3:     ⟨await !inp or last==p⟩;
p4:    critical section;             q4:     critical section;
p5:    inp = false;                  q5:     inq = false;
p6:    noncritical section;          q6:     noncritical section;
       }                                      }
}                                    }
```

a) There is a fatal error in this version. Demonstrate it with a scenario.

b) Correct the problem.

c) Prove it correct by first proving the following as invariants:

Ip: {p3 and q4 => inq and last = p}
Iq: {q3 and p4 => inp and last = q}

And with the help of Ip and Iq by prove that invariance {not (p4 and q4) } is also preserved.

d) Is it possible to replace in p (and symmetrically in q) the atomic ⟨await !inq or last==q⟩ with a busy wait loop where the two parts of the condition are evaluated as independent atomic actions: while(inq and last==p)skip; ? Give a concise argumentation for pro or cons.

3. What are the possible outputs of the following program :

```
semaphore s = 1;
boolean b = false;

process p  {                          process q  {
p1:    P(s);                          q1:    P(s);
p2:    b = true;                      q2:    while not b
p3:    V(s);                          q3:       write ("*");
}                                     q4:    V(s);
                                      }
```

4. The one-lane bridge. Cars coming form north and south have to cross a very long and narrow one-lane bridge. Cars driving to the same direction maybe on the bridge at the same time, but cars heading to opposite directions can't. Consider the following Java 'monitor' code outline for the solution to the problem, where the cars are threads calling the public methods `cross_from_North()` and `cross_from_South()` of the monitor `One_lane_bridge`.

   a) Complete the missing peaces of the synchronization logic of the monitor in Java. Do not worry about the fairness, however the directions should be treated symmetrically. Please be clear when defining the needed variables! Remember that simple, elegant solutions are always the best!
   b) Write down two invariants specifying: 1) There can be several cars on the brigde at the same time, but they must all head to the same direction. 2) There is no unnessary waiting. Give an argument that both of them they are preserved c) Modify your solution to ensure fairness to each direction.

```
class One_lane_bridge {
    ...
    private synchronized void startNorth() {
      ...
    }
    private synchronized void endSouth() {
      ...
    }
    public void cross_from_North() {
      startNorth();
      // north-south crossing operation is embedded here
      endSouth();
    }
    ...
      // south_north crossing is symmetrical
    }
}
```

5. Linda and Tuple Space. a) Write a tuple space solution to the one-lane bridge problem using postnote, removenote, and readnote. b) Modify it to ensure fairness.