1. Consider the following concurrent program. Assume that the underlying hardware provides the normal LOAD and STORE machine instructions so that the program's each individual reference (read or write) to an integer can be considered atomic.

```
int     n = 0;
co      temp1 = n; n = temp1 + 1;      #process p1
//      temp2 = n; n = temp2 + 1;      #process p2
//      while (n<2) print(n);          #process p3
oc
```

Explain you answer to the following question with proper scenarios or proofs. (a) Does the program terminate always? (b) Construct scenarios that print sequences: 012, 002, 02. (c) Must the value 2 appear in the output? (d) How many times can value 2 appear in the output? (e) How many times can value 1 appear in the output?

2. Suppose your machine has the following machine instruction:

```
Swap (var1, var2):
  <  tmp = var1; var1 = var2; var2 =temp;>
```

where `tmp` is a internal register of the processor.

(a) Using `Swap` develop a solution to the critical section problem for n processes. Do not worry about the eventual entry property. Describe clearly how your solution works and why it's correct. (b) Modify your answer to (a) so that it will also perform well with a multiprocessor system with system caches. Explain what changes you have made (if any) and why.

3. Consider the following solution to the one producer – one consumer problem using semaphores and a finite circular buffer. (a) Modify this solution by replacing semaphore operations with busy wait loops and operations referring to the index variables `front` and `rear` only. (b) Enhance this solution so that it will work for several producers and consumer using additional semaphores if necessary.

```
typeT buf[n];                /* an array of some type T*/
int front=0; rear=0;
sem empty=n, full=0;         /* n-2 <= empty+full <=n */

process Producer {
  while (true) {
    ...
    /*produce data, then deposit it in the buffer*/
    P(empty);
      buf[rear]=data;
       rear=(rear+1)%n;
    V (full);
  }}
process Consumer {
  while (true) {
    /*fetch result then consume it */
    P(full);
     result=buf[front];
        front=(front+1)%n;
    V(empty);
    ...
  }}
```

4. The one-lane bridge. Cars coming form north and south have to cross a river along a very long and narrow one-lane bridge. Cars driving to the same direction may be on the bridge at the same time, but cars heading to opposite directions can't. Consider the following generic monitor code outline for the solution to the problem, where the cars are processes calling the public methods `cross_from_North()` and `cross_from_South()` of the monitor `One_lane_bridge`. Complete the missing peaces of the synchronization logic of the monitor procedures for the following cases: (a) Generic monitor without fairness considerations and working both for SC- and SW- semantics. (b) Same as (a) but with Java class and synchronized methods, (c) Your solution should be fair so that cars from each direction will get served in finite time (d) Explain briefly why your solution for (c) works for SC- and/or SW-semantics.

```
monitor One_lane_bridge {
   int ns =0;                  //north-south cars on the bridge
   int sn =0;                  //south-north cars on the bridge
   cond ns_c:                  //condition to enter from north
   cond sn_c;                  //condition to enter from south

   private procedure startNorth() {
     …
     ns++
     }
   private procedure endSouth() {
     ns--;
     …
     }

  public cross_from_North() {  // this is needed to provide a simpler API
    startNorth();
    // north-south crossing operation is embedded here
    endSouth();
  }

 // the south-north direction is symmetric and need not be repeated
}
```

5. Linda and Tuple Space. (a) Explain briefly Linda primitives: OUT, IN, RD, INP, RDP (b) Use them to solve the simple single-use barrier problem for n processes, where the processes have to wait for each other at label p, so that they may only proceed from p, once they all have reached it. (c) Use Linda to solve the five dining philosophers problem. Note that your solutions to both (b) and (c) should be simple, straight-line programs containing just a few lines without loops. (d) Discuss the pro's and con's of using the full power of Linda in this way for (b) and (c).