1 Communication and Termination

The sieve of Eratosthenes computes all prime numbers in a certain range. It works as follows: in order to determine if \( N \) is a prime number, all (previously established) prime numbers smaller than \( N \) are tested in turn and as soon as one divides \( N \) (without remainder) clearly it is not prime so it is discarded. If all primes smaller than \( N \) have been tried without having to discard \( N \), it obviously is a prime number.

A concurrent implementation of this algorithm (which computes the prime numbers up to 100) is given at

http://www.iste.uni-stuttgart.de/fileadmin/user_upload/iste/ps/Lehre/SS2012/V_RTP/a5/sieve.adb

a. Explain how the algorithm works. How many tasks are created and how do they communicate? Show snapshots of the program state to visualise the behaviour of the algorithm.

b. The given implementation does not terminate. Why is that? Rewrite the code so that the program terminates after all prime numbers have been computed.

Hint: You have to add a select statement.
2 Conditional Critical Regions

a. Rewrite the semaphore implementation of the readers/writers problem accessible through the internal university network

http://www.iste.uni-stuttgart.de/fileadmin/user_upload/iste/ps/Lehre/intern/RTP2012/a4/rw/rw_no_starvation.adb

using conditional critical regions instead of semaphores.
(Note: Conditional critical regions are discussed in the lecture notes in section 7.4.1.)

Hint: The solution is much simpler and shorter than the semaphore implementation. It uses only two counters instead of four.

b. Compare conditional critical regions with monitors and Ada protected objects.
(Note: Monitors and protected objects are discussed in the lecture notes in section 7.4.2 and 7.4.3, respectively.)

Briefly outline how the implementation of the readers/writers problem for monitors and Ada protected objects looks like.

3 Protected Objects

Explain the synchronisation behaviour imposed by the following Ada protected object:

```ada
protected type Barrier is
  entry Wait;
private
  Avalanche : Boolean := False;
end Barrier;

protected body Barrier is
  entry Wait when Wait'Count = 42 or Avalanche is
  begin
    if Wait'Count = 0 then
      Avalanche := False;
    else
      Avalanche := True;
    end if;
  end Wait;
end Barrier;
```
The system operates in the following modes:

**Normal mode:** If no intruder is detected, a sensor sends an OK message to the watch
dog every second. If all sensors report an OK message, the watch dog stays calm
and sends an OK message to its successor watch dog in the ring network.

**Alarm mode:** If an intruder is reported (by a sensor), the watch dog closes all entries
and exits, switches on all lights, gives acoustic signals, and calls the GSG9. Fur-
thermore, all other watch dogs are informed of the alarm by sending an alarm
message to the successor watch dog in the ring network. The alarm message is
passed on until it reaches the original sender.
The following events cause a transition from normal mode to alarm mode:

• If a sensor notices an intruder, it immediately goes off and sends its watch dog an alarm message.

• If a sensor has not reported its OK message for 5 seconds, the watch dog sets off the alarm.

• If a watch dog has not received an OK message from its predecessor watch dog for 10 seconds, the watch dog sets off the alarm.

Design a possible solution and write an Ada05 program using tasks (communicating by means of rendezvous) to simulate the described system.
(You can assume that there are 5 watch dogs and 4 sensors for each watch dog.)