**8 Concurrent and Distributed Systems (RNW)**

(*a*) Monitors are a programming primitive linking data with two synchronization types: mutual exclusion and condition synchronisation. Which is provided implicitly; which is provided explicitly? [1 mark]

(*b*) Describe two ways in which Monitors and Conditional Critical Regions differ. [2 marks]

(*c*) The object-oriented programming style encouraged by Monitors has many benefits as the number of data types and locks increases in the system.

(*i*) Placing all data in a single Monitor may improve program correctness. Explain why this might have undesirable performance effects. [1 mark]

(*ii*) One problem that can arise when using multiple locks is *deadlock*, which can be prevented by imposing a partial order on locks. Describe the implications this has for code structure when using Monitors. [2 marks]

(*iii*) Explain why Java's Monitor feature does not necessarily impose this code structure. [2 marks]

(*d*) Condition variables allow condition satisfaction to be signalled between threads. Explain the difference between Hoare's *signal-and-wait* and Mesa's *signal-and-continue* in terms of mutual exclusion and scheduling. [4 marks]

(*e*) Consider the (incorrect) pseudocode on the next page:

(*i*) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Hoare *signal-and-wait* semantics. [4 marks]

(*ii*) Describe and justify minimal modifications to this code, referencing line numbers, in order to make it correct in the presence of Mesa *signal-and-continue* semantics. [4 marks]

[continued . . . ]

```
monitor ProducerConsumer {
  int in, out, buf[N];
  condition notfull, notempty;

  procedure produce(item) {
    if ((in-out) == N)
      wait(notfull);
    buf[in % N] = item;
    if ((in-out) == 0)
      signal(notempty);
    in = in + 1;
  }

  procedure int consume() {
    if ((in-out) == 0)
      wait(notempty);
    item = buf[out % N];
    if ((in-out) == N)
      signal(notfull);
    out = out + 1;
  }

  /* init */ { in = out = 0; }
}
```