```
Handout for CS 372H
Class 11
23 February 2010

1. Readers/writers

    state variables:
        AR = 0;  // # active readers
        AW = 0; // # active writers
        WR = 0; // # waiting readers
        WW = 0; // # waiting writers

        Condition okToRead = NIL;
        Condition okToWrite = NIL;
        Mutex mutex = FREE;

    Database::read() {
        startRead();  // first, check self into the system
        Access Data
        doneRead();   // check self out of system
    }

    Database::startRead() {
        acquire(&mutex);
        while((AW + WW) > 0){
            WR++;
            wait(&okToRead, &mutex);
            WR−−;
        }
        AR++;
        release(&mutex);
    }

   Database::doneRead() {
        acquire(&mutex);
        AR−−;
        if (AR == 0 && WW > 0) { // if no other readers still
          signal(&okToWrite, &mutex);   // active, wake up writer
        }
        release(&mutex);
    }

    Database::write(){  // symmetrical
        startWrite();  // check in
        Access Data
        doneWrite();  // check out
    }

    Database::startWrite() {
        acquire(&mutex);
        while ((AW + AR) > 0) { // check if safe to write.
                                // if any readers or writers, wait
            WW++;
            wait(&okToWrite, &mutex);
            WW−−;
        }
        AW++;
        release(&mutex);
    }

    Database::doneWrite() {
        acquire(&mutex);
        AW−−;
        if (WW > 0) {
            signal(&okToWrite, &mutex); // give priority to writers
        } else if (WR > 0) {
            broadcast(&okToRead, &mutex);
        }
        release(&mutex);
    }

    NOTE: what is the starvation problem here?

```

```
2. Shared locks

    struct sharedlock {
      int i;
      Mutex mutex;
      Cond c;
    };

    void AcquireExclusive (sharedlock *sl) {
      acquire(&sl−>mutex);
      while (sl−>i) {
        wait (&sl−>c, &sl−>mutex);
      }
      sl−>i = −1;
      release(&sl−>mutex);
    }

    void AcquireShared (sharedlock *sl) {
      acquire(&sl−>mutex);
      while (sl−>i < 0) {
        wait (&sl−>c, &sl−>mutex);
      }
      sl−>i++;
      release(&sl−>mutex);
    }

    void ReleaseShared (sharedlock *sl) {
      acquire(&sl−>mutex);
      if (!−−sl−>i)
        signal (&sl−>c, &sl−>mutex);
      release(&sl−>mutex);
    }

    void ReleaseExclusive (sharedlock *sl) {
      acquire(&sl−>mutex);
      sl−>i = 0;
      broadcast (&sl−>c, &sl−>mutex);
      release(&sl−>mutex);
    }

    QUESTIONS:
    A. There is a starvation problem here. What is it? (Readers can keep
       writers out if there is a steady stream of readers.)
    B. How could you use these shared locks to write a cleaner version
       of the code in item 1., above? (Though note that the starvation
       properties would be different.)
```