```
CS3650, week 7.b

The previous handout demonstrated the use of mutexes and condition
variables. This handout demonstrates the use of monitors (which combine
mutexes and condition variables).


1. The bounded buffer as a monitor

  // This is pseudocode. The buffer should be wrapped as an object.
  // Don't take it literally.

  // global buffer info
  int count;
  int in;
  int out;
  Item buffer[BUFFER_SIZE];

  // synchronization variables
  Mutex* mutex;
  Cond* nonempty;
  Cond* nonfull;

  void init()
  {
    in = out = count = 0;
    init(&mutex);
    init(&non_empty);
    init(&nonfull);
  }

  void enqueue(Item item)
  {
    acquire(&mutex);
    while (count == BUFFER_SIZE)
      cond_wait(&nonfull, &mutex);

    buffer[in] = item;
    in = (in + 1) % BUFFER_SIZE;
    ++count;
    cond_signal(&nonempty);
    release(&mutex);
  }

  Item dequeue()
  {
    acquire(&mutex);
    while (count == 0)
      cond_wait(&nonempty, &mutex);

    Item ret = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    --count;
    cond_signal(&nonfull);
    release(&mutex);
    return ret;
  }

```



```
  int main(int, char**)
  {
    init();
    tid1 = thread_create(producer, NULL);
    tid2 = thread_create(consumer, NULL);

    // never reach this point
    thread_join(tid1);
    thread_join(tid2);
    return -1;
  }

  void producer(void*)
  {
    for (;;) {
      /* next line produces an item and puts it in nextProduced */
      Item nextProduced = means_of_production();
      enqueue(nextProduced);
    }
  }

  void consumer(void*)
  {
    for (;;) {
      Item nextConsumed = dequeue();
      /* next line abstractly consumes the item */
      consume_item(nextConsumed);
    }
  }

Key point: *Threads* (the producer and consumer) are separate from
*shared object* (the global buffer). The synchronization happens
in the shared object.


```

```
  2. This monitor is a model of a database with multiple readers and
  writers. The high-level goal here is (a) to give a writer exclusive
  access (a single active writer means there should be no other writers
  and no readers) while (b) allowing multiple readers. Like the previous
  example, this one is expressed in pseudocode.

    // assume that these variables are properly initialized:
    AR = 0; // active readers
    AW = 0; // active writers
    WR = 0; // waiting readers
    WW = 0; // waiting writers

    Condition okToRead = COND_INITIALIZER;
    Condition okToWrite = COND_INITIALIZER;
    Mutex mutex = MUTEX_INITIALIZER;

    void reader() {
      startRead(); // first, check self into the system
      Access Data
      doneRead();
    }

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

    void doneRead() {
      acquire(&mutex);
      AR--;
      if (AR == 0 && WW > 0) { // if no other readers still
        signal(&okToWrite);
      }
      release(&mutex);
    }

    void writer(){ // symmetrical
      startWrite(); // check in
      Access Data
      doneWrite(); // check out
    }

    void 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);
    }

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

  NOTE: what is the starvation problem here?


```