HW 4 Solutions

1.1 Time-of-check-to-time-of-use (TOCTTOU) bugs

The outer "if" is not protected by a mutex. So the interleaving can be:

```
T1: check Bob's balance
T2: decrease Bob's balance
T1: acquire mutex
T1: cause balance to be negative...
```

1.2

Place the `lock()` around the if check. Depending on the implementation, there may need to be two instances of `unlock()` in the code.

2.1 Deadlock

T1 calls `transfer(0, 1, 100)` T2 calls `transfer(1, 0, 100)`

T1: `lock(&mutex[0])` T2: `lock(&mutex[1])` T1: try to `lock(&mutex[1])` T2: try to `lock(&mutex[0])`

2.2

```
bool transfer(int from, int to, double trans) {

        if (from < to) {
            smutex_lock(&mtx[from]);
            smutex_lock(&mtx[to]);
        } else {
            smutex_lock(&mtx[to]);
            smutex_lock(&mutex[from]);
        }

        /*
         * continue as before. for good style, the unlock() should be
         * done in the corresponding order, though that is not
         * required.
         */

    (or, if we know that there are only two mutexes, we could do:
        smutex_lock(&mtx[0]);
        smutex_lock(&mtx[1]);)
}
```

3. Sleeping barber

Waiting Room Solution:

```
Type    Name    Initial Value (if applicable)
-------------------------------------------------------------
mutex   lock
cond    cond
int     nfull               0
int     ticketAvail         0
int     ticketTurn         -1
```

```
int WaitingRoom::enter() {
    lock.acquire();
    int ret;
    if (nfull == NCHAIRS){
      ret = WR_FULL;
    } else {
      ret = MY_TURN;
      myTicket = ticketAvail++;
      nfull++;
      while (myTicket > ticketTurn) {
        cond.wait(&lock);
      }
      nfull--;
    }
    lock.release();
    return ret;
}

int WaitingRoom::callNextCustomer() {
    lock.acquire();
    ticketTurn++;
    if (nfull == 0) {
      ret = EMPTY;
    } else {
      ret = BUSY;
      cond.broadcast();
    }
    lock.release();
    return ret;
}
```

Barber Chair Solution:

```
  Type    Name    Initial Value (if applicable)
 ---------------------------------------------------------------
  mutex   lock
  cond    custUp
  cond    barberGetUp
  cond    sitDown
  cond    seatFree
  cond    cutDone
  int     state           EMPTY
  int     custWalkedIn      0
```

```
void BarberChair::wakeBarber() {
    lock.acquire();
    custWalkedIn = 1;
    barberGetUp.signal(&lock);
    lock.release()
}

void BarberChair::sitInChair() {
    lock.acquire()
    while (state != EMPTY) {
      seatFree.wait(&lock);
    }
    state = LONG_HAIR_CUSTOMER_IN_CHAIR;
    sitDown.signal(&lock);
    while (state != SHORT_HAIR_CUSTOMER_IN_CHAIR) {
      cutDone.wait(&lock);
    }
    state = EMPTY;
    custUp.signal(&lock);
    lock.release();
}

void BarberChair::napInChair() {
    lock.acquire();
    if(custWalkedIn == 0) {   // Cust could arrive before I sit down
      state = BARBER_IN_CHAIR;
    }

    while(custWalkedIn == 0) {
      barberGetUp.wait(&lock);
    }
    custWalkedIn = 0;
    if (state == BARBER_IN_CHAIR) { // Cust could have beaten us
      state = EMPTY
      seatFree.signal(&lock);
    }
```

```
    lock.release();
}

void BarberChair::cutHair() {
    lock.acquire();
    while(state != LONG_HAIR_CUSTOMER_IN_CHAIR) {
      sitDown.wait(&lock);
    }
    state = SHORT_HAIR_CUSTOMER_IN_CHAIR;
    cutDone.signal(&lock);
    lock.release();
}

void BarberChair::tellCustomerDone() {
    lock.acquire();
    while(state != EMPTY){  // NOTE: No other cust can arrive until I call call_next_cust()
      custUp.wait(&lock);
    }
    lock.release();
}
```