Question 1 – Monitor pseudo-code

At a minimum you need to:

- track how many customers are in the shop,
- allow customers to wait in line until their turn to get a haircut,
- allow the barber to sleep when there are no customers, and be woken by a customer arriving to an empty shop, and
- let a customer wait until notification from the barber that their haircut is done.

A simple solution has the barber always go to sleep after finishing a haircut; if there are any waiting customers, he will be woken immediately:

```plaintext
int nCustomers
condition Barber, Waiting, Done

barber:
loop:
wait Barber
<cut hair>
signal Done

10 if nCustomers < 5:
   nCustomers++
7 if nCustomers > 1:
   wait Waiting
3

customer:
if nCustomers < 5:
   nCustomers++
6

11 if sleeping:
   signal Barber
9

wait Done

12 if nCustomers--
8 signal Waiting
13
```

Note that after signaling the customer (line 5) the barber will always go to sleep (line 3) before the customer wakes on line 11. Thus if there is another customer waiting, it will be signaled (line 13) and wake from line 9 after the barber went to sleep. This is important – if by any chance the next customer could signal before the barber went to sleep, the signal would be “lost”, and the barber would remain wedged on line 3.

A slightly more complicated version has the barber only go to sleep if there are no waiting customers, and a custom wake the barber only if he is sleeping:

```plaintext
int nCustomers
bool sleeping
condition Barber, Waiting, Done

barber:
loop:
if nCustomers == 0:
   sleeping = True
4
   if sleeping:
5      signal Barber
6
   wait Barber
7
   sleeping = False
8
   <cut hair>
9
   signal Done
10 if nCustomers < 5:
   nCustomers++
11 if nCustomers > 1:
   wait Waiting

12 if nCustomers--
13 signal Waiting
```

Note that there is a harmless data race here – if customer 1 sees sleeping=True and signals the barber, the barber might not wake and set 'sleeping' to False before customer 2 arrives, resulting in an extra call to signal(Barber) on line 15.
Question 2 – Pthreads implementation

Here's a direct translation of the first monitor implementation above:

```c
int nCustomers;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t Barber = PTHREAD_COND_INITIALIZER;
pthread_cond_t Done = PTHREAD_COND_INITIALIZER;
pthread_cond_t Wait = PTHREAD_COND_INITIALIZER;

void barber_method(void) {
    pthread_mutex_lock(&mutex);
    while (1) {
        pthread_cond_wait(&Barber, &mutex);
        double t = exponential(0.8);
        mon_sleep(t, &mutex);
        pthread_cond_signal(&Done);
    }
    pthread_mutex_unlock(&mutex);
}

void customer_method(int i) {
    pthread_mutex_lock(&mutex);
    if (nCustomers >= 5)
        printf("%f: customer %d: full shop\n", mon_now(), i);
    else {
        nCustomers++;
        if (nCustomers > 1)
            pthread_cond_wait(&Wait, &mutex);
        pthread_cond_signal(&Barber);
        printf("%f: customer %d: sitting in barber chair\n", mon_now(), i);
        pthread_cond_wait(&Done, &mutex);
        printf("%f: customer %d: haircut done\n", mon_now(), i);
        nCustomers--;
        pthread_cond_signal(&Wait);
    }
    pthread_mutex_unlock(&mutex);
}
```

It really doesn't have to be any more complicated than this.

Note that most of my testing of question 2 involved running it for a while at 100x speedup to see if I hit any race conditions.
Question 3 – Simulation

Now we just make a few changes (mon_now → sim_now, mon_sleep or sleep → sim_sleep) and add code to collect statistics. Here we use the code provided in stat.c:

```c
int nCustomers;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t Barber = PTHREAD_COND_INITIALIZER;
pthread_cond_t Done = PTHREAD_COND_INITIALIZER;
pthread_cond_t Wait = PTHREAD_COND_INITIALIZER;
int total, turnedAway;
struct cc_stat inShop;
struct id_stat waitTime;
void barber_method(void) {
    pthread_mutex_lock(&mutex);
    while (1) {
        pthread_cond_wait(&Barber, &mutex);
        double t = exponential(0.8);
        mon_sleep(t, &mutex);
        pthread_cond_signal(&Done);
    }
    pthread_mutex_unlock(&mutex);
}
void customer_method(int i) {
    pthread_mutex_lock(&mutex);
    total++;
    double t0 = sim_now();
    if (nCustomers >= 5)
        turnedAway++;
    else {
        cc_event(&inShop, nCustomers);
        nCustomers++;
        if (nCustomers > 1)
            pthread_cond_wait(&Wait, &mutex);
        pthread_cond_signal(&Barber);
        pthread_cond_wait(&Done, &mutex);
        pthread_cond_signal(&Wait);
        cc_event(&inShop, nCustomers);
        ic_event(&waitTime, sim_now() - t0);
        nCustomers--;
    }
    pthread_mutex_unlock(&mutex);
}
...  
printf("turned away: %d of %d (%f)\n", turnedAway, total,
1.0*turnedAway/total);
printf("mean wait time: %f\n", id_mean(&waitTime));
for (i = 0; i < 6; i++)
    printf("%d in shop: %f\n", i, cc_fract(&inShop, i));
... 
```

If you use my statistics functions, you have to remember to sample the number of customers in the shop before you increment or decrement it, rather than afterwards. Otherwise it's pretty straightforward.
**Question 4 – Markov Model**

The continuous-time Markov model for this system is shown to the right, with states numbered according to the total number of customers in the barbershop. (i.e. in the barber chair or any of the 4 waiting chairs)

The transition rate from any state $S_i$ to $S_{i-1}$ is the rate at which the barber completes haircuts, or 0.8. The transition rate from $S_i$ to $S_{i+1}$ depends on how many customers are outside the barbershop, as each potential customer wakes up with a rate of 0.1 and tries to enter the shop, for a total transition rate from $S_i$ to $S_{i+1}$ of $0.1(10-i)$.

A) Setting up the balance equations:

$$0.8 \cdot P_5 = 0.6 \cdot P_4$$

$$(0.8+0.6)P_4 = 0.8 \cdot P_5 + 0.7 \cdot P_3$$

$$(0.8+0.7)P_3 = 0.8 \cdot P_4 + 0.8 \cdot P_2$$

$$(0.8+0.8)P_2 = 0.8 \cdot P_3 + 0.9 \cdot P_1$$

$$(0.8+0.9)P_1 = 0.8 \cdot P_2 + 1.0 \cdot P_0$$

$$P_5 + P_4 + P_3 + P_2 + P_1 + P_0 = 1$$

(note that the last balance equation, equating the input and output rates of $P_0$, is redundant so we omit it. In fact, any one of the state balance equations could be chosen to be omitted.)

First we do it the hard way, by back substitution:

$$P_4 = \frac{4}{3} P_5$$

$$P_3 = \frac{1}{7} (14 P_4 - 8 P_5) = \frac{32}{21} P_5$$

$$P_2 = \frac{1}{8} (15 P_3 - 8 P_4) = \frac{32}{21} P_5$$

$$P_1 = \frac{1}{9} (16 P_2 - 8 P_3) = \frac{256}{189} P_5$$

$$P_0 = \frac{1}{10} (17 P_1 - 8 P_2) = \frac{1024}{945} P_5$$

$$P_5 = \frac{945}{7389} = 0.012789$$

$$P_4 = \frac{1260}{7389} = 0.17052$$

$$P_3 = \frac{1440}{7389} = 0.19488$$

$$P_2 = \frac{1280}{7389} = 0.17323$$

$$P_1 = \frac{1024}{7389} = 0.13858$$
Now the easy way:
\[
\begin{bmatrix}
0.8 & -0.6 & 0 & 0 & 0 & 0 \\
-0.8 & 1.4 & -0.7 & 0 & 0 & 0 \\
0 & -0.8 & 1.5 & -0.8 & 0 & 0 \\
0 & 0 & -0.8 & 1.6 & -0.9 & 0 \\
0 & 0 & 0 & -0.8 & 1.7 & -1.0 \\
1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
P_5 \\
P_4 \\
P_3 \\
P_2 \\
P_1 \\
P_0
\end{bmatrix} =
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
1
\end{bmatrix}
\]
or \( A \cdot P = Y \) and \( P = A^{-1} \cdot Y \)

Setting this up in matlab or octave:

octave:1> A = [0.8,-0.6,0,0,0,0; -0.8,1.4,-0.7,0,0,0; 0,-0.8,1.5,-0.8,0,0; 0,0,-0.8,1.6,-0.9,0; 0,0,0,-0.8,1.7,-1; 1,1,1,1,1]
A =
0.80000  -0.60000   0.00000   0.00000   0.00000   0.00000
-0.80000   1.40000  -0.70000   0.00000   0.00000   0.00000
0.00000  -0.80000   1.50000  -0.80000   0.00000   0.00000
0.00000   0.00000  -0.80000   1.60000  -0.90000   0.00000
0.00000   0.00000   0.00000  -0.80000   1.70000  -1.00000
1.00000   1.00000   1.00000   1.00000   1.00000   1.00000
octave:2> Y = [0;0;0;0;0;1]
Y =
0
0
0
0
0
1
octave:3> inv(A) * Y
ans =
0.12789
0.17052
0.19488
0.19488
0.17323
0.13858

B) What is the probability of a customer turning away due to a full shop?

This isn't just P5, the probability that there are 5 customers in the shop, as the arrival rate changes with the number of customers in the shop. Instead we augment the Markov model above with the 5-5 transition corresponding to customers arriving to a full shop:

Then we calculate the rate across this transition - \( 0.5 \cdot P5 = 0.06395 \) - and compare that to the total arrival rate,
\[
0.5 \cdot P5 + 0.6 \cdot P4 + 0.7 \cdot P3 + 0.8 \cdot P2 + 0.9 \cdot P1 + P0 = 0.75306
\]
giving us
\[
\frac{0.06395}{0.75306} = 0.084913
\]

C) What is the expected time in the shop (including haircut) if the shop is not full?

Again we have to adjust for the varying arrival rates, as explained in class on 2/23. We know that the expected waiting time is \( \frac{i+1}{0.8} \) for a customer arriving when there are \( i \) customers already in the shop. We can now calculate the expected waiting time per unit time by looking at the arrival rates into
each state:

\[
0.6 \, P_4 \, \frac{5}{0.8} + 0.7 \, P_3 \, \frac{4}{0.8} + 0.8 \, P_2 \, \frac{3}{0.8} + 0.9 \, P_1 \, \frac{2}{0.8} + 1.0 \, P_0 \, \frac{1}{0.8} = 2.4692
\]

Now we divide by the arrival rate of customers who *are not turned away*:

\[
0.6 \, P_4 + 0.7 \, P_3 + 0.8 \, P_2 + 0.9 \, P_1 + 1.0 \, P_0 = 0.68912
\]

and we get our final answer, 3.58307