Name: ____________________________

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing.

SHOW YOUR WORK!

1. (20) In the example of the population of three people, p.161, find the probability that $\overline{X}$ overestimates the population mean μ. Give your answer in a common fraction, simplified to the lowest terms.

2. (20) State the mailing tube used in line 14 of the code on p.166, citing a specific equation number.

3. (30) The following code finds the true confidence level for a certain kind of confidence interval. Fill in the gaps. Note: The optional argument **prob** in **sample()** gives sampling probabilities. If for instance we wish to simulate one roll of a trick die that has probability 0.25 of a 6 and probabilities 0.15 each for 1,2,3,4,5, the call would be **sample(1:6,1,prob=c(0.15,0.15,0.15,0.15,0.15,0.25)**.

```
simpci <- function(m,k,r) {
   contain <- vector(length=m)
   for (i in 1:m) {
      samp <- sample(                       )          # gap 1
      center <-                                        # gap 2
      rad <- 1.96 * sqrt(center*(1-center)/k)
      contain[i] <-                                    # gap 3
   }
   print(mean(contain))
}
```

4. (30) Here you will use **DES.R** to simulate a *mirrored* file server. There are two disks with identical contents, assumed here to be read-only. Read requests arrive to the server according to a Poisson process with intensity parameter v. The track number of a request is uniformly distributed on (0,1). The read/write head of a disk travels at speed s during seeks, so that the time to traverse all tracks is 1/s. After a seek, if no other requests are pending, the read/write head of a disk stays at the track of its last seek. For simplicity, assume that when a request is fulfilled by a disk and other requests are waiting, this disk will be the one to serve the next request, even if the other disk is idle and its current track is closer. Fill in the gaps. Note: **rep()** does a "repeat" operation, so that for instance **rep(5,3)** is (5,5,5).

```
dsim <- function(totsimetime,v,s) {
   initglbls(v,s)
   ...
   return(totmovtime/(2*totsimetime))
}

initglbls <- function(v,s) {
   arvrate <<- v  # arrival rate
   seekspd <<- s  # seek speed
   # last known positions of the 2 disks, in (0,1)
   lastpos <<- rep(0.0,2)
   # destinations of the 2 disks (if moving), in (0,1)
   nextpos <<- rep(0.0,2)
   # booleans for currently idle
   idle <<- rep(TRUE,2)
   # start times of the current moving times of the 2 disks
   # (if moving), changing from an idle state
   startmov <<- rep(0.0,2)
   # number of waiting requests
   q <<- 0
   arvtype <<- 1  # arrival type
   seektype <<- 2  # seek done type
   totmovtime <<- 0
}

reactevnt <- function(head) {
   if (head[2] == arvtype) {  # arrival
      ...
   } else {  # seek done
      disknum <- head[3]
      lastpos[disknum] <<- nextpos[disknum]
      if (q > 0) {
         nextpos[disknum] <<-                          # gap 1
         q <<-                                         # gap 2
         seektime <-                                   # gap 3
         seekdonetime <- sim$currtime + seektime
         schedevnt(seekdonetime,seekdonetype,disknum)
      } else {
         totmovtime <<-                                # gap 4
                                                       # gap 5 (a
                                                            # full line)
      }
   }
}
```

Solutions:

1. The population mean $\mu = (69 + 70 + 72)/3 = 70.33$. In (6.3), we see that the possible values of $\overline{X}$ above this value are 70.5, 71 and 72, which collectively have probability 5/9.

2.a The proper mailing tube is (6.17); (3.30) is also OK.

3. The point of this problem was to gauge how well you understood the meaning of confidence intervals, especially pp.168-169, and also the notion of indicator variables. This is why the arguments to **simpci()** were not definied; your ability to infer them from context was a measure of your understanding of those concepts. Here is the full code:

```
simpci <- function(m,k,r) {
   contain <- vector(length=m)
   ctrs <- vector(length=m)
   for (i in 1:m) {
      samp <- sample(0:1,k,replace=T,prob=c(1-r,r))
      center <- mean(samp)
      ctrs[i] <- center
      rad <- 1.96 * sqrt(center*(1-center)/k)
      contain[i] <- (abs(center-r) < rad)
   }
   print(mean(contain))
}
```

4. Here is the full code, including for the "..." portions:

```
dsim <- function(totsimetime,v,s) {
   initglbls(v,s)
   # create simulation
   newsim()
   # get things going, generating and scheduling first arrival event
   arvtime <- rexp(1,rate=arvrate)
   schedevnt(arvtime,arvtype,arvtime)
   mainloop(totsimetime)
   return(totmovtime/(2*totsimetime))
}

initglbls <- function(v,s) {
   # globals
   # rates
   arvrate <<- v  # arrival rate
   seekspd <<- s  # seek speed (time 1/s to traverse all tracks)
   # last known positions of the 2 disks, in (0,1)
   lastpos <<- rep(0.0,2)
   # state variables
   # destinations of the 2 disks (if moving), in (0,1)
   nextpos <<- rep(0.0,2)
   # booleans for currently idle
   idle <<- rep(TRUE,2)
   # start times of the current moving times of the 2 disks (if moving),
   # changing from an idle state
   startmov <<- rep(0.0,2)
   # number of queued requests
   q <<- 0
   # event types
   arvtype <<- 1  # arrival type
   seektype <<- 2  # seek done type
   # statistics
   totmovtime <<- rep(0.0,2)
}

reactevnt <- function(head) {
   if (head[2] == arvtype) {  # arrival
      if (idle[0]) disknum <- 0
      else if (idle[1]) disknum <- 1
      else disknum <- -1
      if (disknum == -1) q <<- q + 1
      else {  # process the request
         idle[disknum] <<- FALSE
         # generate destination
         nextpos[disknum] <<- runif(1)
         seektime <- abs(nextpos[disknum]-lastpos[disknum]) / seekspd
         seekdonetime <- sim$currtime + seektime
         schedevnt(seekdonetime,seekdonetype,disknum)
      }
      # generate next arrival
      arvtime <- sim$currtime + rexp(1,arvrate)
      schedevnt(arvtime,arvtype)
      startmov[disknum] <<- sim$currtime
   } else {  # seek done
      disknum <- head[3]
      lastpos[disknum] <<- nextpos[disknum]
      if (q > 0) {
         nextpos[disknum] <<- runif(1)
         q <<- q - 1
         seektime <- abs(nextpos[disknum]-lastpos[disknum]) / seekspd
         seekdonetime <- sim$currtime + seektime
         schedevnt(seekdonetime,seekdonetype,disknum)
      } else {
         totmovtime <<- sim$currtime - startmov[disknum]
         idle[disknum] <<- TRUE
      }
   }
}
```