MCS 360 Programming Project #2

In this problem, you are to perform a simulation involving events that occur randomly in time. Consider the following situation. The numbers λ_a and λ_s are positive real constants – each the λ of a Poisson distribution – and n is a positive integer constant.

> A bank is open for 8 hours (480 minutes) each day, say from 9:00 a.m. until 5:00 p.m. The bank has n tellers. Customers arrive at the bank at independent random times, according to a Poisson distribution, but the average rate at which they arrive is λ_a customers per minute. The mean time between arrivals is $1/\lambda_a$ minutes, and the time between arrivals follows an exponential distribution: $prob(time\ between\ arrivals \le t) = 1 - e^{-t/\lambda_a}$. In a simulation, if one customer arrives at time t, we may choose the arrival time of the next customer as $t - \ln(1-r)/\lambda_a$, where r is pseudo-random in [0,1). Of course, we must choose a new pseudo-random r for each customer.
>
> When a customer arrives, if one or more tellers are idle (not serving other customers), the customer goes immediately to the window of some idle teller (say chosen randomly) and is served without waiting. Otherwise the customer must wait. Our goal is too compare several strategies for operating the bank, which will determine how customers wait. Eventually (before the bank closes), a waiting customer is served.
>
> The time that it takes the teller to serve an individual customer varies according to a Poisson distribution, but on average a teller can serve λ_s customers per minute, although the customer may serve fewer customers because he/she is idle. Thus the mean time to serve a customer is $1/\lambda_s$ minutes, and the service time follows an exponential distribution: $prob(service\ time \le t) = 1 - e^{-t/\lambda_s}$. One may simulate the service time of a customer as $-\ln(1-r)/\lambda_s$, where r is pseudo-random in [0,1). Of course, we must choose a new pseudo-random r for each customer.
>
> Once a teller has finished serving a customer, that customer leaves the bank. After the closing time (5:00 p.m.), no more customers are allowed to enter the bank. However, customers already inside the bank at closing time will be served. The bank actually closes when the last customer leaves, or at 5:00 p.m., whichever is later.

In this situation, we may be interested in questions such as:

i) How does the number of customers served vary from day to day? (We expect this number to average $480\lambda_a$, but it will vary from day to day.)

ii) During a typical day, how many customers are served immediately? How many must wait? What is the greatest amount of time that any customer waits to be served during the day? What is the mean waiting time among those customers forced to wait? How do these numbers vary from day to day?

iii) During a typical day, for how long is the teller idle?

v) Approximately what time does the bank actually close? How long after 5:00 p.m.?

Below are four strategies for operating the bank, that will affect the answers to these questions. Strategy 2' is quite realistic, but it is rather difficult to program. Your program should implement strategies 1, 2, and 3.

1) [*multiple lines, no line switching, each teller serves only his/her line*] There is a separate line (a queue) for each teller. When a customer enters the bank and no teller is available, the customer enters a shortest queue. (In case of a tie, the customer could choose randomly among shortest queues.) Once entering a queue, the customer waits in

that queue until served. When any teller becomes available, the teller immediately begins to serve the customer at the front of that teller's particular queue. If the teller's queue is empty, then the teller remains idle (or performs other tasks at the bank) until a new customer enters the bank and chooses his/her window.

> *Note:* This strategy isn't particularly efficient, nor is it particularly fair, since a new customer entering the bank may be served immediately while customers already waiting continue to wait. But it does approximate what would occur if we had *n* separate banks, each with one teller, rather than one bank with *n* tellers.

2) [*multiple lines, no line switching*, *each teller serves his/her line preferentially*] This is like (1) above, except that when any teller becomes available, the teller immediately begins to serve the customer at the front of his/her particular queue. However, if the teller's queue is empty, then the teller immediately begins to serve the customer at the front of another teller's queue, choosing randomly among the nonempty queues. If all queues are empty, the teller remains idle until another customer enters the bank and chooses his window.

> *Note:* A strategy similar to this is common used, though in practice some line switching may occur.

2') [*multiple lines, line switching permitted, each teller serves his/her line preferentially*] This is like (2) above, except that a customer is allowed to change lines. However, a customer switching from his/her line to another line must go to the rear of the new line. This mean that the k^{th} customer in some line will (most likely) switch only to a line with $k-2$ or fewer customers. Even then, not all customers switch lines. To simulate this strategy, we would need estimates of the likelihood that a customer will in fact change lines, and how conflicts are resolved when more than one customer would like to switch to the same line.

3) [*a single line*] There is a single waiting line (a queue). When a customer enters the bank and no teller is available, the customer goes to the rear of this single queue. Whenever any teller becomes available, that teller immediately begins to serve the customer at the front of the single queue. If the single queue is empty, the teller remains idle until another customer enters the bank.

> *Note:* This strategy is commonly used in practice. An equivalent strategy is for entering customers to take a number (given out in sequence). When a teller becomes available, that teller begins serving the waiting customer with the lowest number.

Your program should read in five numbers:

n: [an `int`] The number of tellers.

λ_a: [a `double`] The average number of customers arriving at the bank in one minute.

λ_s: [a `double`] The average number of customers that a teller can serve in one minute.

d: [an `int`] The number of days for which the simulation should run.

seed: [an `int`] The seed used to initialize the random number generator. The random number generator should be initialized just once, at the start of your program.

The your program should simulate the operation of the bank for d days under each of the three strategies. For each of the three strategies (1,2,4), for each day (0,1,...,$d-1$), it should print nine numbers on a line, in fields of fixed width.

i) The strategy number (an `int`, equal 1, 2, or 3).
ii) The day number (an `int`, equal 0, 1, 2, ..., $d-1$).
iii) The number of customers arriving at the bank between 9:00 and 5:00 that day (an `int`).
iv) The number of customers served immediately upon entering the bank (an `int`).
v) The number of customers forced to wait for service (an `int`).
vi) The average waiting time, during the day, of customers forced to wait (a `double`).
vii) The longest waiting time, during the day, of any waiting customer (a `double`).
viii) The combined idle time, during the day, of all tellers (a `double`).
ix) The number of minutes after 5:00 that the bank actually closes (a `double`).

The numbers of type `double` can be written with one digit after the decimal point. Once this data has been written out, it can be imported into an Excel spreadsheet, making it easy to compute means, inter-quartile ranges, and standard deviations of (iii) through (ix) above.

This program should be submitted by Wednesday, October 31.