"List processing allows one to manipulate records without physically moving them in storage."

Discrete System Simulation

by Armando de la Cruz*

Introduction

Simulation is a problem solving technique. However, it does not generate alternative solutions. Rather, it only evaluates alternative operating policies and system configurations. Furthermore, simulation by itself cannot provide the best operating policy or the best system configuration.

Simulation attempts to duplicate or imitate the behavior of a system over time through the use of a dynamic model. Simulation therefore provides decision makers a basis for improving system performance through operating procedure modification or system redesign.

Systems

A system is a set of interacting objects, each having its own characteristics. We refer to the objects as system entities and to the characteristics of an object as attributes. The interactions among the objects which are called activities cause changes in the system.

The set of attributes of an entity defines the state of that entity, and the states of the critical entities of a system define the state of that system. The study of system behavior over time involves the study of its state changes as time elapses. An event is said to have occurred whenever there is a change in the state of an entity. There is therefore no change in the states of entities between events. Hence, simulation is a process of making all state changes at the time corresponding to a particular event and then moving simulated time to the time of the next event.

Applications

Simulation has been applied to a wide variety of problem areas. Listed below are examples of some of its applications.

1. Simulation of the operations of a supermarket to determine system performance as affected by changes in policies (e.g., number of open

* Professor of Industrial Engineering, U.P. College of Engineering

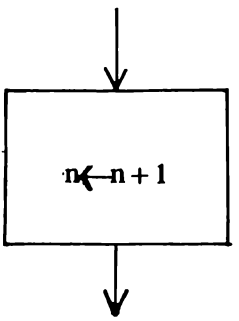
checkout counters, number of express checkout counters, maximum number of items a customer should have for him to be allowed to join the express checkout counter, and so on).

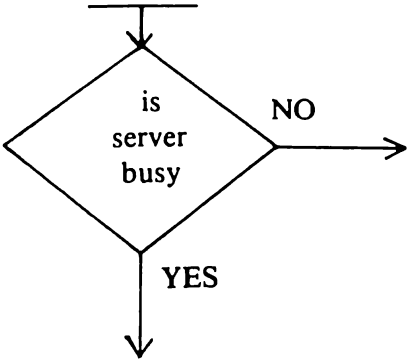
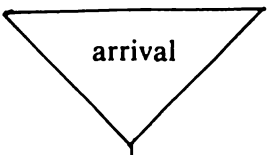
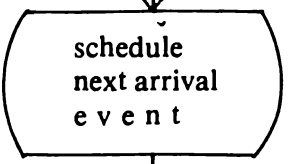
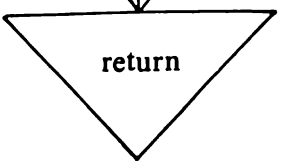
2. Simulation of the operations of a bank to test policies designed to shorten customer waiting time.
3. Simulation of a pedestrian crossing to determine acceptable time duration of walk time.
4. Simulation of a job shop to determine the amount of in-process storage space that should be provided in each department.
5. Simulation of a job shop to compare alternative dispatching policies.
6. Simulation of steel-making operations to evaluate alternative sizes of oxygen tank and oxygen delivery rate.
7. Simulation of a maintenance operation to determine the best size of repair crews.
8. Simulation of a city street lamp system to determine the best replacement policy.
9. Simulation of an inventory system of a factory to evaluate alternative reordering policies.

Model Structure

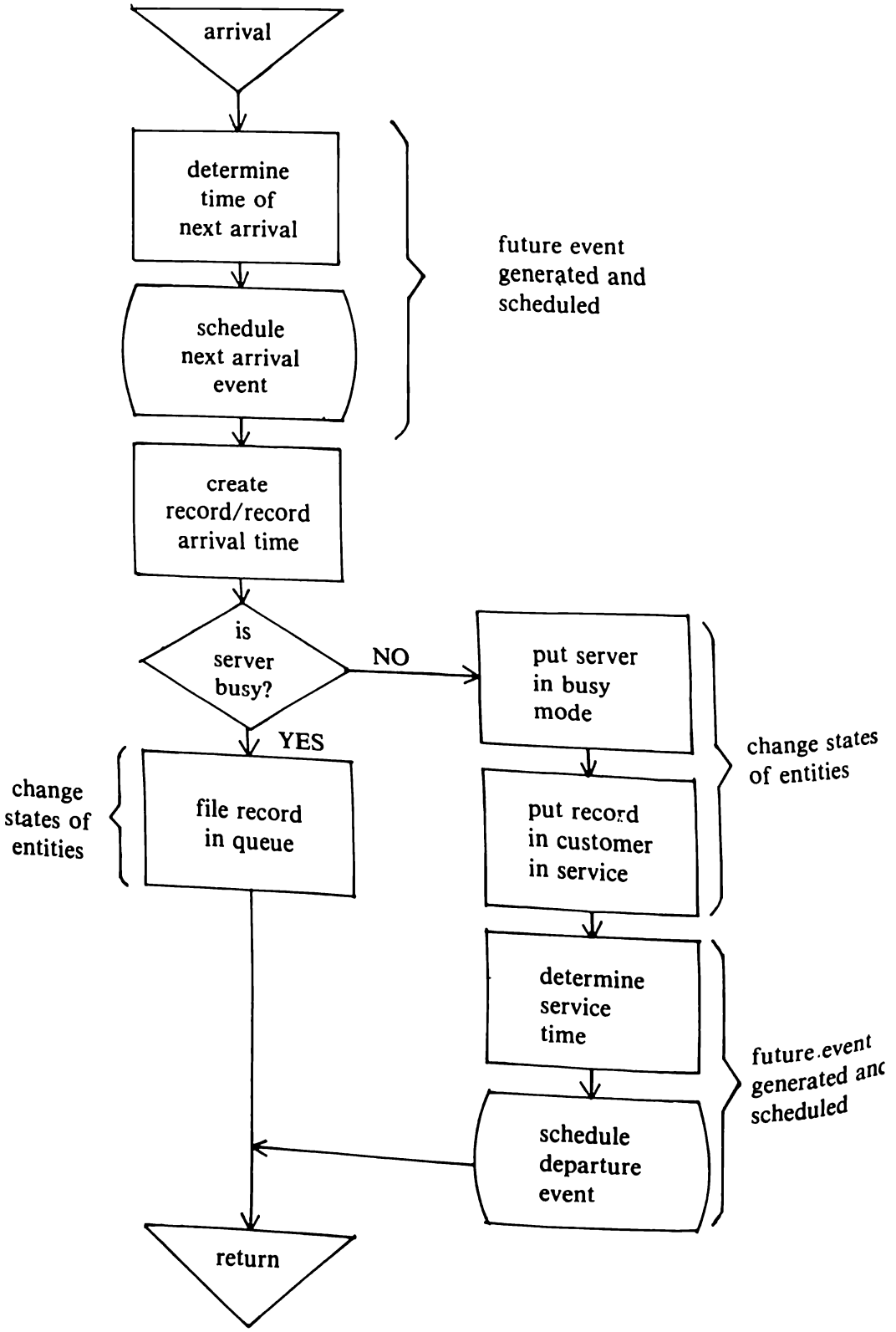
Two sets of relationships are used in representing a system for simulation. One set includes the mathematical relationships that exist between attributes associated with the entities. For example, if n is the number of customer in a system, we set n to (n + 1) when a customer arrives and we set n to (n — 1) when a customer departs. The other set includes the logical relationships that exist between conditions (states) and actions. For example, when s server becomes free and there are other jobs waiting (conditions) he begins service (action). If no job is waiting (condition) he remains free (action).

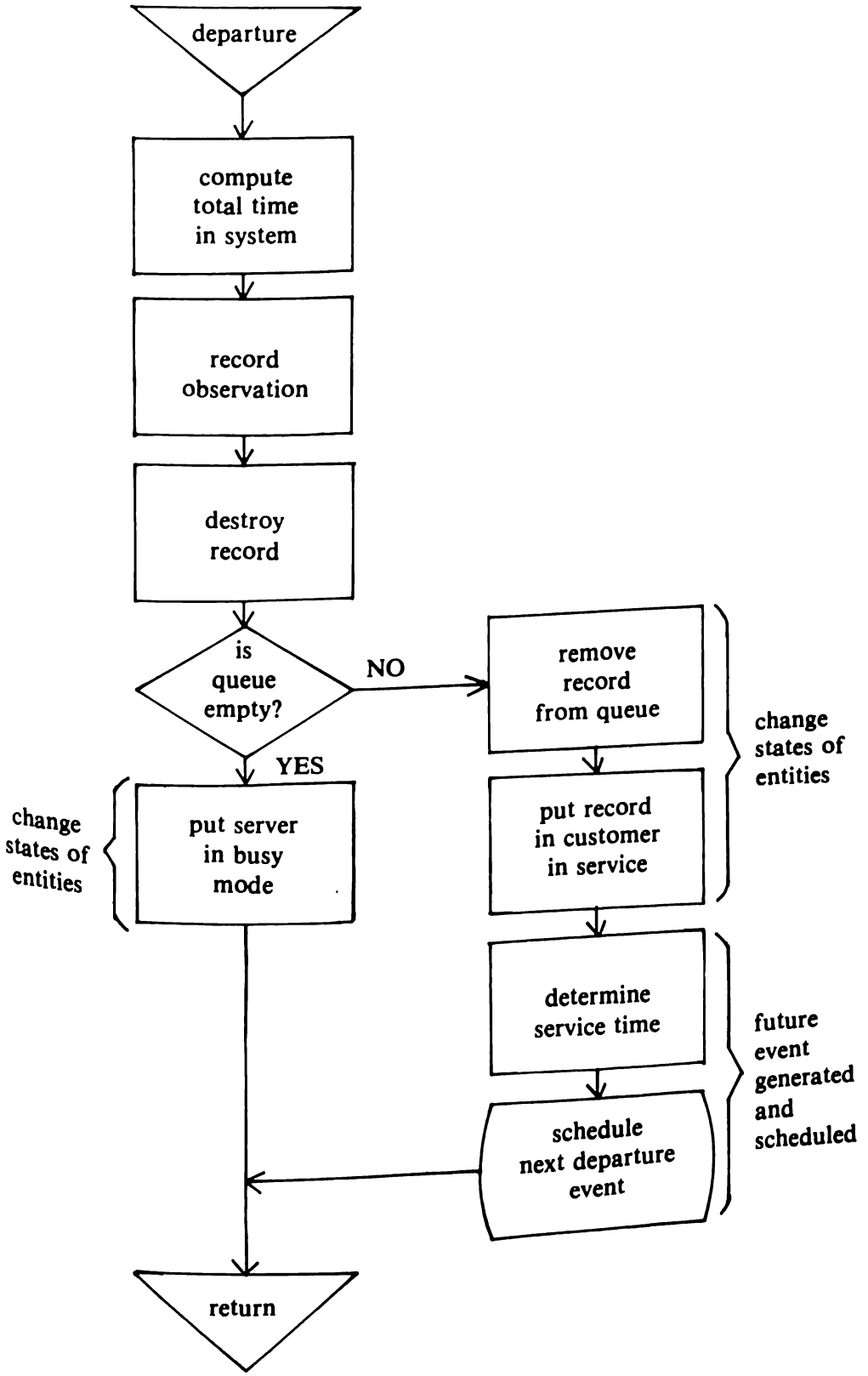
A convenient way of describing the interactions among the entities is through the use of a flowchart. One may use the conventions given below:

Name	*Symbol*	*Meaning*
box		operation

Name	*Symbol*	*Meaning*
diamond		decision flow
triangle		flowchart entry
drum		schedule a future event
triangle		flowchart exit

Consider a simple queueing system where cars arrive at a service station with one pump. There are two events here—arrival of a car and departure of a car. Using then the symbols given above we describe an arrival event and a departure event.


arrival
determine time of next arrival
schedule next arrival event
future event generated and scheduled
create record/record arrival time
is server busy?
NO
YES
put server in busy mode
change states of entities
file record in queue
change states of entities
put record in customer in service
determine service time
future event generated and scheduled
schedule departure event
return

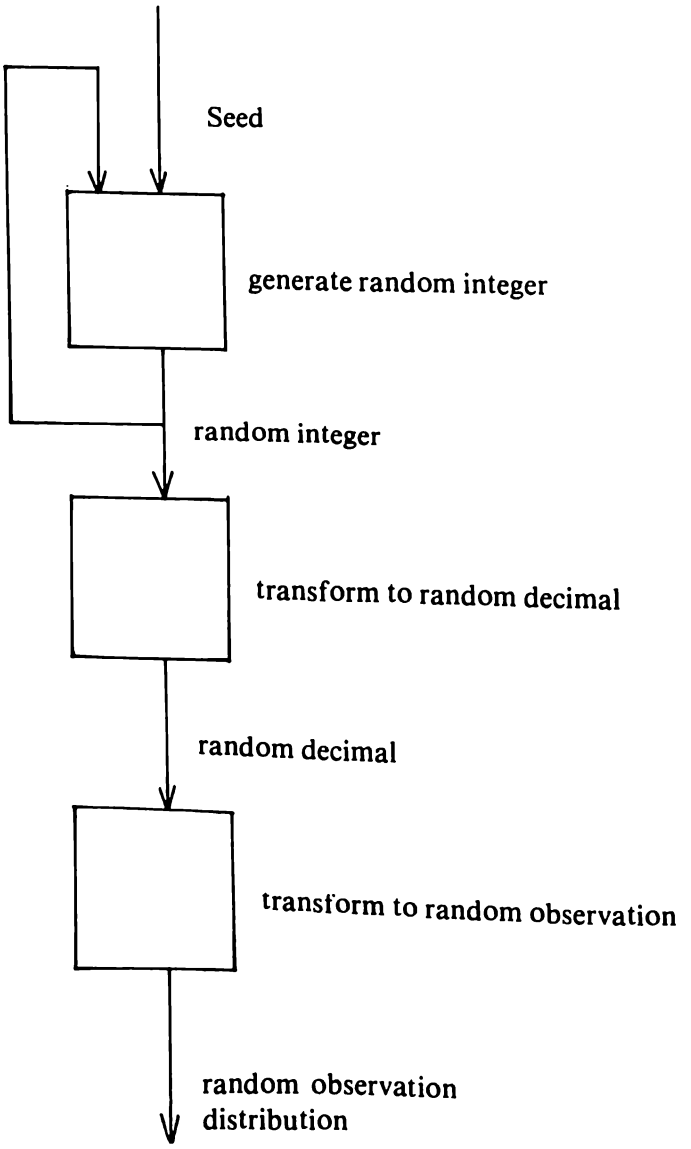

departure
compute total time in system
record observation
destroy record
is queue empty?
NO
YES
remove record from queue
put record in customer in service
change states of entities
change states of entities
put server in busy mode
determine service time
schedule next departure event
future event generated and scheduled
return

Notice that each event accomplishes two things. First, future events are generated and scheduled. Second, changes on the states of entities are made.

Observe also that time is frozen when an event is executed. It may take sometime to execute an event from entry to exit but current time remains unchanged.

Determination of next event time

The generation of further events is a process of three operations as shown below:



A random integer may be generated using the equation

$$Z_i = a\,Z_{i-1} + c - \left[\frac{a\,Z_{i-1} + c}{m}\right] x\ m$$

where [*] denotes the largest integer in *.

Z_0 is referred to as the seed of the generator.

As an example, let a = 5, c = 0, Z_0 = 4, and m = 7. Then we have the following integer generation:

i	computation	Z_i
0		4
1	20 — (20/7) x 7	6
2	30 — (30/7 x 7	2
3	10 — (10/7) x 7	3
4	15 — (15/7) x 7	1
5	5 — (5/7) x 7	5
6	25	—
		(2
6	25 — (25/7) x 7	4

Then the number Z_i is transformed to U_i by

$$U_i = Z_i/m$$

i	Z_i	U_i
0	4	
1	6	0.8571
2	2	0.2857
3	3	0.4286
4	1	0.1429
5	5	0.7143
6	4	0.5714

Lastly, U_i is transformed to a random observation from a probability distribution by first getting the cumulative distribution function from the probability density function and then equating this to U_i.

Suppose that T has the probability density function

$$f_T(t) = \begin{cases} \dfrac{1}{b-a}, & a \leq t \leq b \\ 0, & \text{elsewhere} \end{cases}$$

Then

$$F_T(t) = \int_a^t \frac{ds}{b-a}$$

$$= \frac{t-a}{b-a}$$

So we have

$$U_i = F_T(t_i) = \frac{t_i - a}{b-a}$$

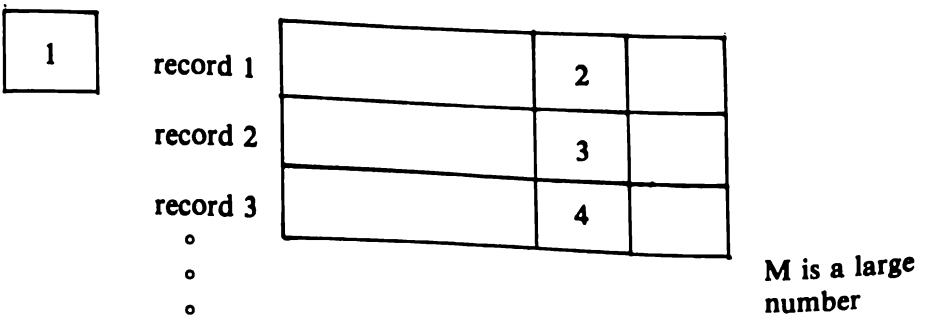
or

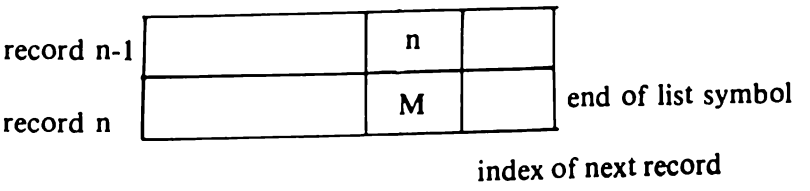
$$T = t_i = U_i(b-a) + a$$

List Processing

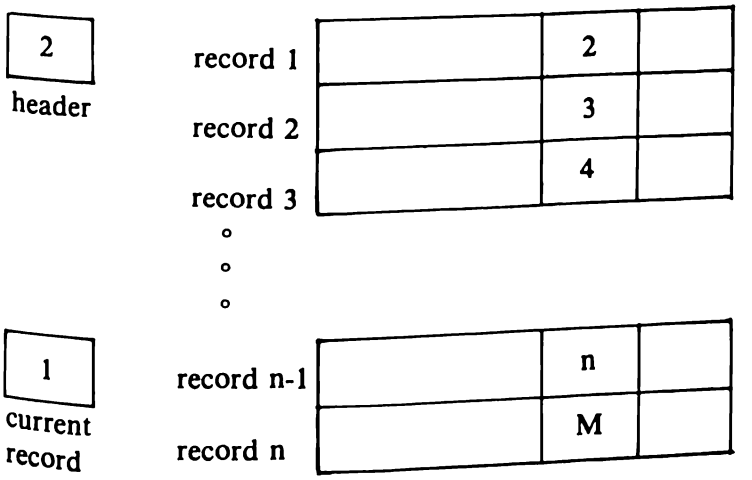
A substantial part of simulation involves record manipulation. It is therefore important to make use of a programming technique known as list processing. This technique allows one to manipulate records without physically moving them in storage. This technique is illustrated below for processing three types of lists — a first-in-first-out (FIFO) list, a last-in-first-out (LIFO) list, and a priority list.

A stack of records may be structured as a LIFO list. When a record is created, the first record in the list is removed from the list. When a record is destroyed, the record is returned to the list at the head of the list. Each record in the list contains the index of the next record on the list. This is called a pointer. A variable, called a header, contains the index of the first record in the list. The last record in the list contains an end of list symbol in its pointer. If the list is empty the header contains the end of list symbol.

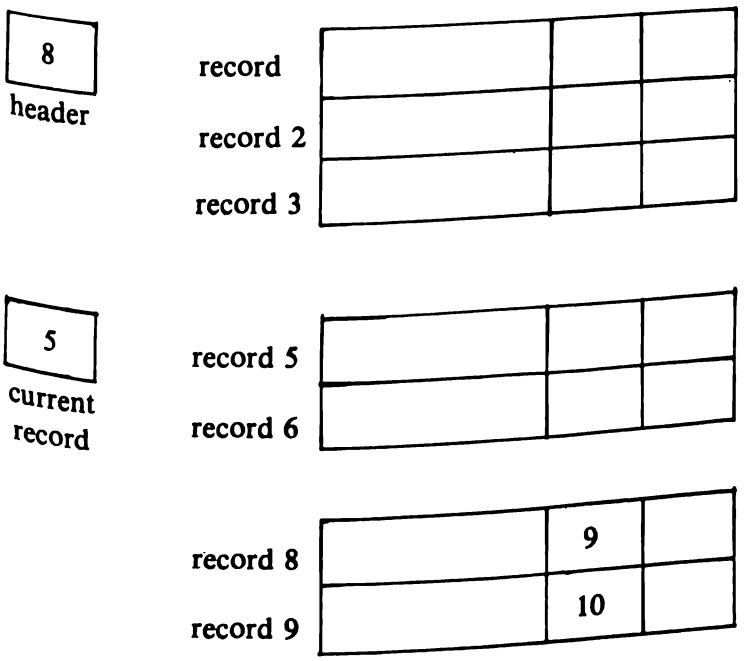


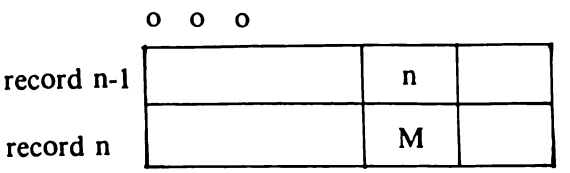


To create a record, the variable current record gets the value in the header, and the header gets the value of the record addressed to by the header.

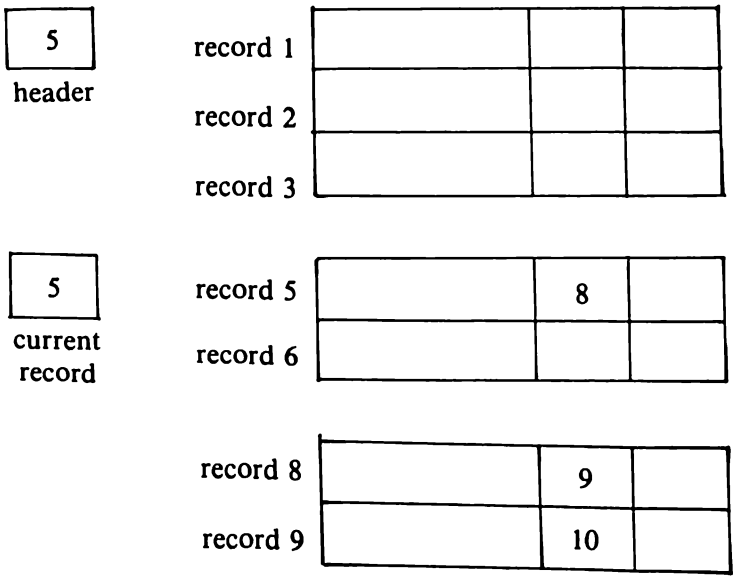


To destroy a record, the pointer of the current record gets the value in the header, and the header gets the value in the current record. Suppose before record 5 is destroyed, the stack looks

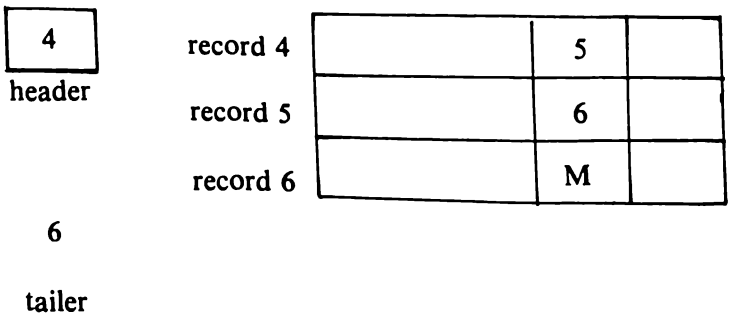




After destroying record 5, the stack would look

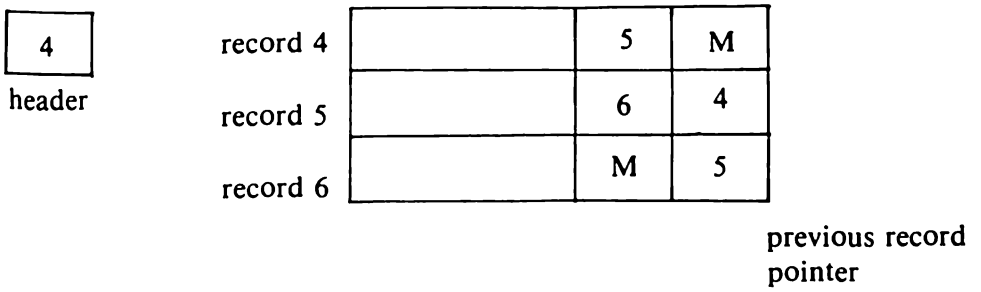


In a FIFO list, it is convenient to have another variable called a tailer that contains the index of the last record in the list. This makes it easy to add a record at the end of the list. Suppose record 6 is the last record in the list. The list would look



To add a record at the end of the list, the pointer of the record addressed to by the tailer gets the value of the current record, the tailer gets the value of the current record, and the pointer of the current record gets the end of list symbol.

In a priority list, it is convenient to add another pointer in each record. This points to the previous record in the list.



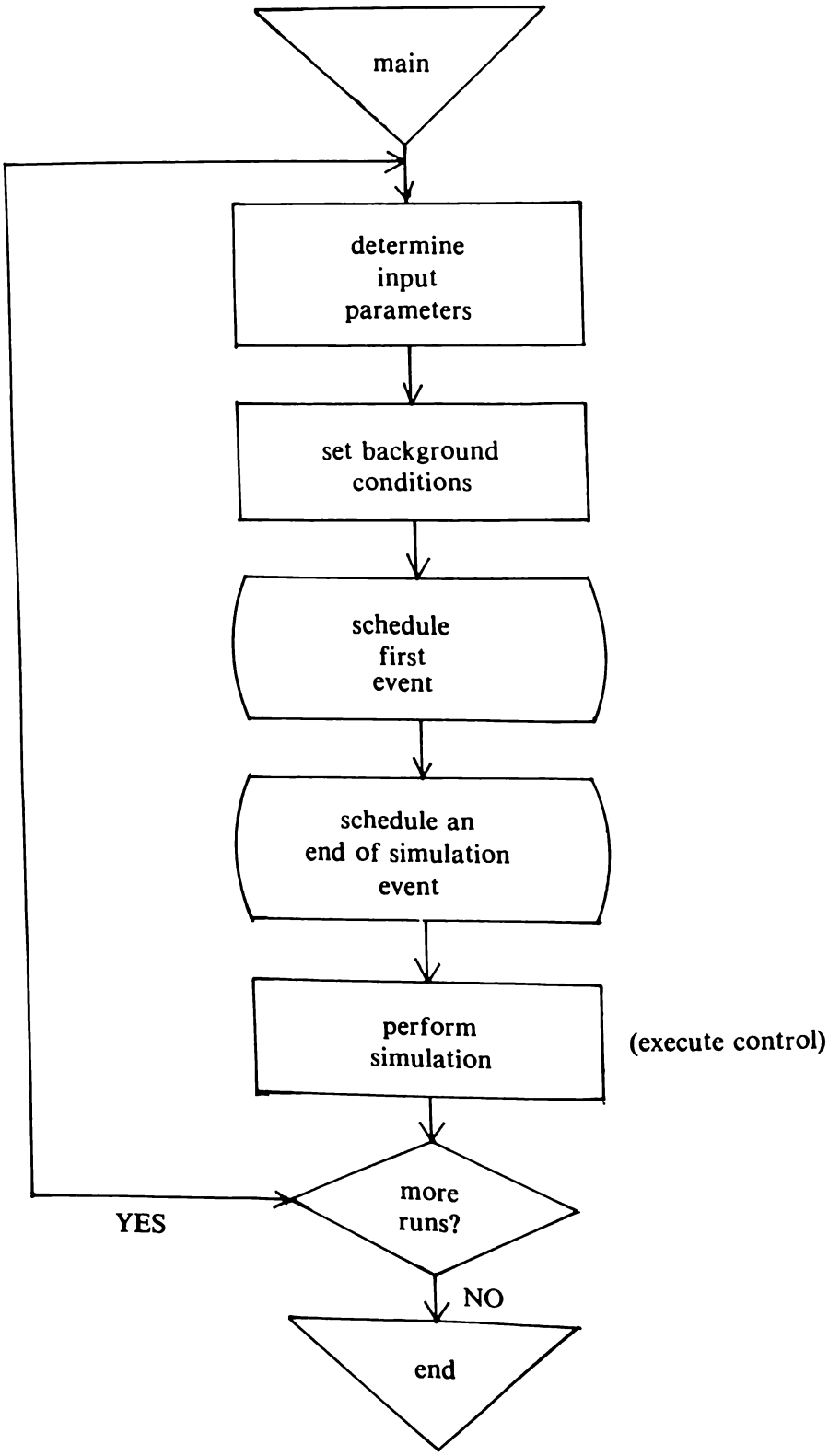
To add a record before a specific record in the list, the next record pointer of the current record gets the index of the specific record. Then the next record pointer of the record previous to the specific record gets the value of the current record. Also the previous record pointer of the current record gets the value of the previous record pointer of the specific record. Lastly, the previous record pointer of the specific record gets the value of the current record.

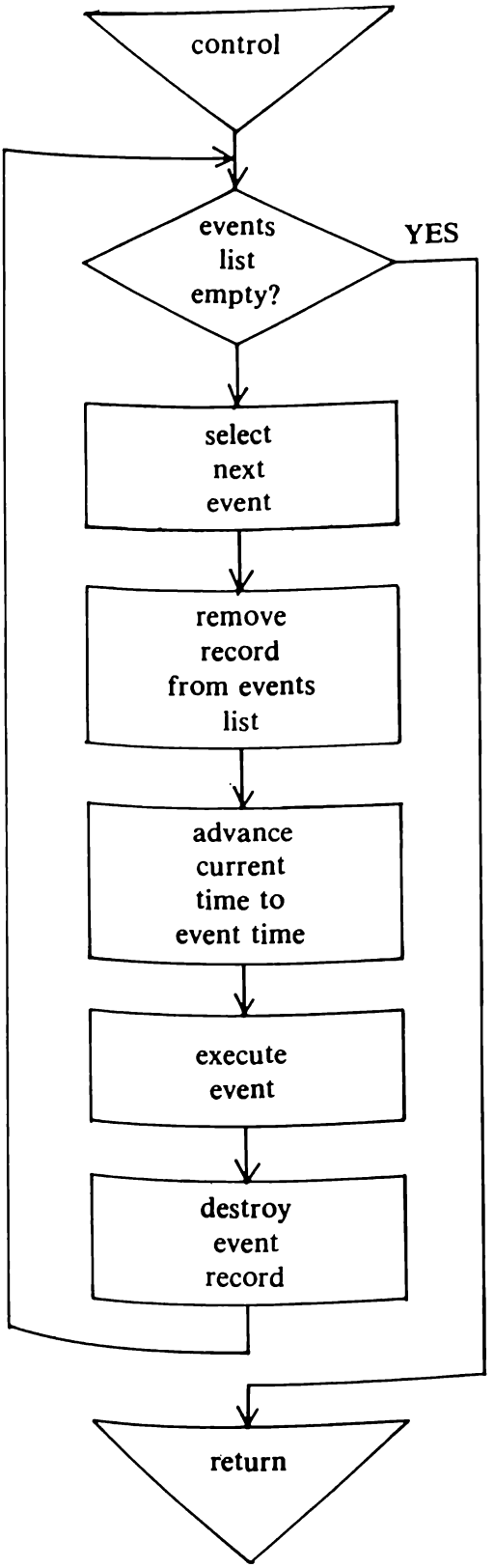
Running the Simulation

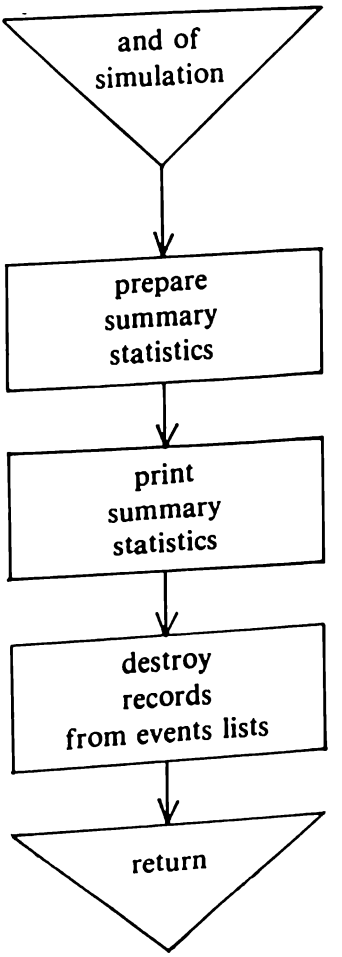
A simulation program may have a four-level structure—the main program, the control program, the event modules, and the support functions such as generation of random observations, list processing, and computation of mathematical functions.

The main program performs initialization and determines the number of simulation runs. The control program provides the timing routine and transfers execution from event to event.

The following diagrams show sample main, control, and an end of simulation event programs.


main
determine input parameters
set background conditions
schedule first event
schedule an end of simulation event
perform simulation
(execute control)
more runs?
YES
NO
end


control
events list empty?
YES
select next event
remove record from events list
advance current time to event time
execute event
destroy event record
return


and of simulation
prepare summary statistics
print summary statistics
destroy records from events lists
return

References

1. G.S. Fishman, *Concepts and Methods in Discrete Event Digital Simulation.* New York: John Wiley and Sons, 1973.

2. G. Gordon, *The Application of GPSS V To Discrete System Simulation.* New Jersey: Prentice-Hall, 1975.

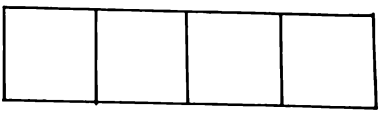
3. F.S. Hillier and G.J. Lieberman, *Introduction to Operations Research.* San Francisco: Holden-Day, 1980.

4. H.G. Daellenback and J.A. George, *Introduction to Operations Research Techniques.* Boston: Allyn and Bacon, 1978.

5. W.G. Graybeal and U.W. Pooch, *Simulation: Principles and Methods.* Cambridge: Winthrop Publishers, 1980.

Data Structures

1. event record

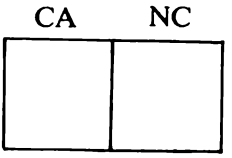


CE = current event record
KE = event code
NT = time of occurrence of event
NE = next event record
PE = previous event record

event code

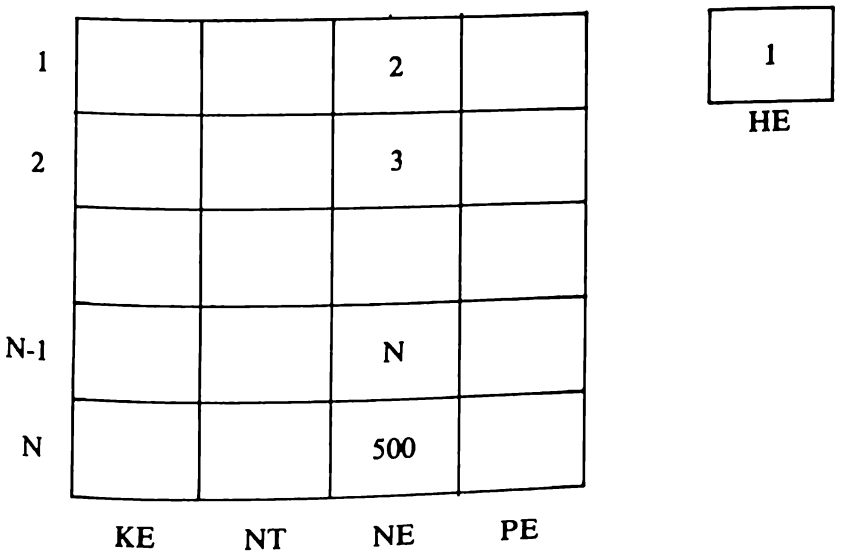
1 denotes arrival event
2 denotes departure event
3 denotes end of simulation event

2. customer record



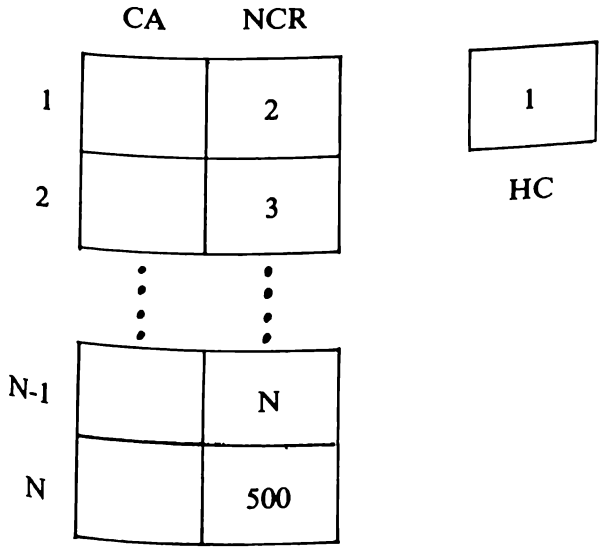
CC = current customer record
CA = customer arrival time
NC = next customer record

3. event records stack



HE = head of event records stack
NE (I) = 500 denotes record I is the last record
N = initial size of stack
HE = 500 denotes stack is empty

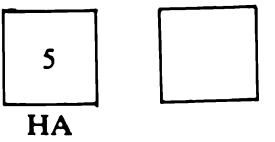
4. customer records stack



HC = head of customer records stack
NC (I) = 500 denotes record I is the last record
N = initial size of stack
HC = 500 denotes stack is empty

5. events list

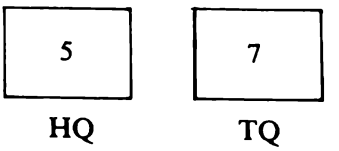
	KE	NT	NE	PE
5	1	10	10	500
10	1	20	15	5
15	2	25	17	10
17	3	6000	500	15



HA = head of events list
NE (I) = 500 means I is the last record
PE (I) = 500 means I is the first record
HA = 500 means events list is empty
CA = index of record that comes after a record that is to be inserted

6. customer queue

	CA	NC
5	10	6
6	12	7
7	20	500



HQ = head of customer queue
TQ = tail of customer queue
NC (I) = 500 means record I is the last record
HQ = 500 means the queue is empty
TQ = 500 means the queue is empty

7. server

MO = denotes status of server

MO = 1 denotes server is busy
MO = 2 denotes server is idle.


CS

CS = index of customer record that is in service

8. other variables time

S1 % = seed for interval between arrivals
S2 % = seed for service times
Z % = random integer
u = random decimal
T % = random observation
M = means of observation
CL(I) = class interval I
LE = time of occurrence of end of simulation event
NN = size of event records stack
NM = size of customer records stack
KL = clock
KO = event code
M1 = mean of time intervals between arrivals
M2 = mean of service times
LA = events list sentinel
LA = 1 means event record to be added at the tail of the list
LA = 2 means event record to be added in location other than the tail of the list
CT = total time in system of a customer
CM = the index of current event record

```
10 REM   MAIN PROGRAM
20 REM
30 REM   THIS PROGRAM DEMONSTRATES THE SIMULATION
40 REM   OF A SINGLE SERVER QUEUEING SYSTEM.
50 REM
60 REM   THE INPUTS TO THIS PROGRAM ARE SIZE OF EVENT
70 REM   RECORDS STACK, SIZE OF CUSTOMER RECORDS STACK,
80 REM   LENGTH OF SIMULATION RUN, MEAN INTERARRIVAL
90 REM   TIME, AND MEAN SERVICE TIME. THE OUTPUT OF THIS
100 REM   PROGRAM IS THE FREQUENCY DISTRIBUTION OF
110 REM   TOTAL TIME A CUSTOMER SPENDS IN THE SYSTEM.
130 REM
140 CLS
150 REM   DECLARE ARRAYS AND VARIABLES
160 DEFINT C,H,I,K,L,M,N,P,T
```

```
170 DIM   KE(100), NT(100), NE(100), PE(100), CA(100), NC(100), CL(11)
180 REM
190 REM   INITIALIZE SERVER STATUS, CUSTOMER QUEUE,
200 REM   EVENTS LIST, SEEDS, AND CLASS ARRAY
210 MO= 2
220 HQ= 500
230 TQ= 500
240 HA= 500
242 FOR I=1 TO 11
243 CL(I)= 0
244 NEXT I
245 S1%= 159
246 S2%= 283
247 INPUT "MEAN INTERARRIVAL TIME(<=100)"; M1
248 IF M1>100 OR M1<0 THEN 247
249 INPUT "MEAN SERVICE TIME(<=70)"; M2
250 IF M2>70 OR M2<0 THEN 249
260 IF M2>=M1 THEN 249
262 REM
264 REM
264 REM   CREATE RECORD STACKS
266 INPUT "SIZE OF EVENT STACK(20<=SIZE<=100)"; NN
270 IF NN>100 OR NN<20 THEN 266
280 INPUT "SIZE OF CUSTOMER STACK(20<=SIZE<=100)"; NM
290 IF NM>100 OR NM<20 THEN 280
300 N= NN
310 GOSUB 2000
320 N= NM
330 GOSUB 3000
340 CLS
350 PRINT "EVENT STACK CREATED. SIZE= "; NN
360 PRINT "CUSTOMER STACK CREATED. SIZE= "; NM
370 REM
400 REM   SCHEDULE AN ARRIVAL EVENT
410 KL= 0
420 T%= 0
430 KO= 1
440 GOSUB 5000
450 CLS
460 PRINT "ARRIVAL EVENT SCHEDULED AT CLOCK TIME= "; KL
470 REM
480 REM   SCHEDULE AN END OF SIMULATION EVENT
490 INPUT "LENGTH OF RUN(500<=LENGTH<=6000)"; T%
500 IF T%>6000 OR T%<500 THEN 490
510 KL=0
520 KO= 3
```

```
530 GOSUB 5000
540 CLS
550 PRINT "END OF SIMULATION EVENT SCHEDULED AT"
560 PRINT "CLOCK TIME = "; T%
570 REM
580 REM   EXECUTE SIMULATION
590 GOSUB 800
600 END
800 REM   SUBROUTINE SIMULATE
810 REM
820 REM   THIS SUBROUTINE PERFORMS THE TIMING ROUTINE
830 IF HA = 500 THEN RETURN
832 REM
840 REM   GET FIRST RECORD FROM EVENTS LIST
850 CM = HA
860 HA = NE(HA)
862 PE(HA) = 500
870 REM
880 REM   UPDATE CLOCK TIME TO EVENT TIME
890 KL = NT(CM)
900 REM
910 REM   EXECUTE EVENT
920 ON KE(CM) GOSUB 6000, 7000, 8000
930 REM
940 REM   DESTROY EVENT RECORD
942 CE = CM
950 GOSUB 1500
960 GOTO 830
1000 REM SUBROUTINE FOR GENERATING RANDOM DECIMAL
1010 V = 2*Z% + 271
1020 Z% = V—(V/10000)*10000
1030 U = Z%/10000
1040 RETURN
1500 REM SUBROUTINE TO DESTROY RECORD OF EVENT
1510 NE(CE) = HE
1520 HE = CE
1530 NN = NN + 1
1540 RETURN
2000 REM SUBROUTINE TO CREATE STACK OF EVENT RECORDS
2010 FOR I = 1 TO N—1
2020 NE (I) = I + 1
2030 NEXT I
2040 NE(N) = 500
2050 HE = 1
2060 RETURN
3000 REM SUBROUTINE TO CREATE CUSTOMER RECORDS
```

```
3010 FOR I = 1 TO N—1
3020 NC(I) = I + 1
3030 NEXT I
3040 NC(N) = 500
3050 HC = 1
3060 RETURN
4000 REM SUBROUTINE TO GENERATE RANDOM OBSERVATION
4010 T% = —M*LOG(1-U)
4020 RETURN
4500 REM SUBROUTINE TO CREATE CUSTOMER RECORD
4510 CC = HC
4520 HC = NC(HC)
4530 NM = NM—1
4540 RETURN
```