SIMSCRIPT notes

[ACKNOWLEDGEMENT: The following teaching notes have material taken from the SIMSCRIPT (CACI, Inc) user manual and other SIMSCRIPT documents. ]

SIMSCRIPT language is both a general-purpose programming language and a specialized simulation programming language. As a programming language, it is English-like, free-form, modular. Separate compilation. Dynamic storage.

For learning purposes, it may be thought of as having five levels:
Level I-III. General programming constructs.
Level IV. Data Structures which can be used for many types of programs but are particularly important to simulation. Entities, Attributes, Sets.
Level V. Simulation programming. Events. Time advance mechanisms, stochastic variables.
FREEFIELD INPUT

READ A, B, C

treats input as a
continuous stream
of data.

Data: 12.3 15.18 0

START NEW RECORD

comments:
not double quote
2 single quotes

ex
READ D "DAY", M "MONTH", Y "YEAR"

FREEFIELD OUTPUT

LIST start - built-in format

PRINT 1 LINE WITH A, B, C THUS...
A = xxxxx, B = xxx, C = .xxx

if A is integer, will print xxx000
if B is real, will round, not truncate

PRINT 2 LINES THUS
REPORT
A  B  SUM  DIFFERENCE
Statement Label:
   single quote
   'HERE' LET A = B
   GO TO 'HERE' (these quotes not necessary)

output (c+d):

To print wide line (beyond input record length):

PRINT 1 DOUBLE LINE THUS...
[1 ← 74)
(75 (1) ← 132 (58)]

Skip 5 Lines
Start New Page

Physical "end" of program:

END

Logical end of program
"SIMPLESCRIPT" Programming

The problem is to evaluate a banking system which is organized such that all arriving customers join a single queue if all the tellers are busy. Then, when a teller becomes free, the customer at the head of the queue goes to that teller for service.

We give three parameters to characterize our system:

\( \lambda \) : the average arrival rate of customers
\( \mu \) : the average service rate of tellers
\( k \) : the number of tellers on duty in the bank

The measurements we desire are:

\( L_q \) : the average queue length
\( L \) : the average number in the system
\( W_q \) : the average waiting time
\( W \) : the average time in system

If we assume that everything is perfectly random in our system, then we assume that arrivals are distributed according to the Poisson distribution and service times are distributed exponentially. Then, results may be obtained analytically, using the following formulas derived from queueing theory:

\[
\rho = \frac{\lambda}{\mu}
\]

\[
L_q = \frac{\rho^{k+1}}{(k-1)! (k-\rho)^2} P_0
\]

\[
W = W_q + \frac{1}{\mu}
\]

\[
P_0 = \frac{1}{\left(\sum_{n=0}^{k-1} \frac{\rho^n}{n!}\right) + \frac{\rho^k}{(k-1)! (k-\rho)}}
\]

\[
W_q = \frac{L_q}{\lambda}
\]

\[
L = L_q + \rho
\]

Write a "SIMPLESCRIPT" program to input values for \( \lambda \) and \( \mu \), and to print the values of \( L, L_q, W, W_q \) for varying values of \( k \). Your program should make note of the fact that solutions are valid only if \( \lambda/(k \mu) < 1 \).

Use the following data:

\( \lambda = 20 \) arrivals/hour
\( \mu = 10 \) customers/hour
\( k \) varying from 1 to 5 by steps of 1
**SELECTED LEVEL 3 TOPICS**

**SEARCHING**:

A very important facility for simulation. To search a group of values for the first meeting some criterion:

\[
\text{FOR } i = 1 \text{ TO } N, \\
\text{WITH GRADES}(i) > 3 \\
\text{FIND THE FIRST CASE} \quad [\text{or, } \text{FIND } i \text{ = } \text{THE FIRST } i] \\
[\text{or, } \text{FIND } \text{var } = \text{arith. exp.}] \\
\]

When using this search facility, two additional operators on the IF statement are available: IF FOUND and IF NONE (i.e., no value of \( i \) meets the specified conditions).
SUMMARY STATISTICS

COMPUTE compute-list OF arith. exp.

The compute-list is a list of phrases of the form: variable = statistic name or variable AS THE statistic name. For list of statistics, see Table 3-1 on p.168.

Suppose we have collected some data values (e.g., exam scores) and stored them in an array called X. Then,

FOR I = 1 TO 40, WITH X(I) > 0 AND X(I) < 100,
COMPUTE AVG.GRADE AS THE MEAN,
VAR.GRADE AS THE VARIANCE,
S.D.GRADE AS THE STD.DEV,
MIN.GRADE AS THE MINIMUM, AND
MAX.GRADE AS THE MAXIMUM OF X(I)
Routines

Preamble: data declaration section

```
PREAMBLE

END
```

Main Routine: only one

```
MAIN

SUBPROGRAM

END
```

Subprogram:

simplest is ROUTINE

```
ROUTINE SQUARE, ROOT

RETURN

END
```

Subprogram is executed by CALL statement.

```
ROUTINE name

RETURN

END
```

<logical end

<physical end
GLOBAL, LOCAL Variables
values are passed to + from subprograms in 2 ways:
- implicitly, as values of global variables, or
- explicitly, via argument lists

In FORTRAN, use COMMON stmt to define a GLOBAL variable
in PL/I, default is global (e.g. var is unless defined as local)
In SIMSCRIPT, it's the opposite - default is local.

to define a variable as global:
PRE AMBLE

DEFINE X AS A VARIABLE
END

Now X is a global variable. It is defined for and accessible by all subroutines unless there
is a DEFINE stmt in the subroutine itself.
ROUTINE ARGUMENTS:

Using routine arguments makes transmission of values between routines explicit, rather than implicit (i.e., by using global variables). In this way, the logical relationship between the routines is made clear and the chance of error is reduced.

There are two kinds of routine arguments:

"GIVEN" arguments, with values supplied by the calling routine, and "YIELDED" arguments (initially set to zero), with values which are transmitted to the calling routine.

ROUTINE name GIVEN argument list 1 YIELDING argument list 2

In PREAMBLE:

DEFINE name AS [A] ROUTINE [GIVEN i ARGUMENTS] [YIELDING j ARGUMENTS] GIVING VALUES WITH
ROUTINES AS FUNCTIONS:

A function is a routine which returns only a single value. The CALL statement is a bit "heavy" for this kind of routine.

--- The subprogram code itself is basically the same as for any routine, except that the RETURN statement must indicate the value to be returned: RETURN WITH arithmetic expression. Also, the word ROUTINE may be replaced with the word FUNCTION.

--- In order to "call" the function, the name of the function subprogram is simply inserted into the expression in place of the "yielded" value: LET X = SQUARE.ROOT(NUMBER).

--- In the PREAMBLE: DEFINE name AS mode FUNCTION.

Library Functions:

*Use* them!! See Appendix B.
Some Library Functions

ABS_F (e) = |e|
EXP_F (e) = e^e
EXPONENTIAL_F (e_1, e_2) = a random sample from an exponential dist. e_1 = mean, e_2 = a stream
INT_F (e) = rounded integer portion of e
MAX_F (e_1, e_2, ..., e_n) = the largest e_i
MIN_F (e_1, e_2, ..., e_n) = the smallest e_i
NORMAL_F (e_1, e_2, e_3) = a random sample from a normal dist. e_1 = μ, e_2 = σ, e_3 = a stream
RANDOM_F (e) = a random # between 0 & 1, e = a stream
SQR_F (e) = \sqrt{e}
TRUNC_F (e) = truncated integer portion of e
DEFINE / TO MEAN

global, local variables

DEFINE RHO TO MEAN LAMBDA / MU.
SIMSCRIPT DATA STRUCTURES

THE PRIMITIVE ELEMENTS ARE:

TEMPORARY ENTITIES
(LEADING TO PROCESSES)

PERMANENT ENTITIES
(LEADING TO RESOURCES)

SETS
(COLLECTIONS OF ENTITIES)
**Subscripted Variables:**
- dynamic storage allocation
- the same storage space may be reserved for 2 different arrays that we don't use at the same time.
- array does not have to be rectangular; may be:
  
  ![Image of a ragged array]

  
  In PREAHOLE: DEFINE A AS A 2-DIM ARRAY.

  In execution of program: a pointer variable
  RESERVE A(*,*) AS 3 BY 4.

  A(*,*) points to A(1,*), etc.
  A(1,*) points to A(1,1), etc.
  A(3,*) points to A(3,1), etc., i.e., the individual values of A

  = READ A rightmost subscript varies fastest
  = RELEASE A space cleared
  = RESERVE #(*,*) AS 10 BY 20.
RESERVE X(\(*\), \*) AS 5 BY X

FOR I = 1 TO 5, DO...
    READ M
    RESERVE X(I, \*) AS M
    FOR J = 1 TO M READ X(I, J)

Loop

X(\(*, \*)
X(1, \*)
X(2, \*)
...
X(5, \*)

<table>
<thead>
<tr>
<th>x(11)</th>
<th>...</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>

This bunch initialized to X.

As long as there as an \* in the reference, it is a pointer, not a variable (value).

LET Y(\*) = 0 is done on RELEASE.
LEVEL 4 - DATA STRUCTURES

Suppose a variable called STUDENT had three characteristics: STUDENT.NO, GPA, AND CLASS.STANDING. We could set up a 2-dimensional array to store and manipulate this information. Then, STUDENT(2,11) would represent the grade point average of the 11th person. This approach works, but it is cumbersome, hard to understand, and prone to error. Wouldn't it be better if we could access this information as GPA(STUDENT(11))? 

We will view the world as a series (or, perhaps, as collections) of entities and the attributes associated with each entity. These may be temporary entities (e.g., jobs) or permanent entities (e.g., machines). A set is a collection of entities. At any given moment in time, the state of the system may be described by: the number of entities, the attributes of each entity, and the set membership of each entity.
ENTITIES, ATTRIBUTES:

The entity, STUDENT, is defined in the Preamble by:

EVERY STUDENT HAS A STUDENT.NO, A GPA, AND A CLASS.STANDING

In general,

EVERY entity-name HAS AN attribute-name-list

Synonyms for AN: A, SOME, THE
Synonyms for HAS: MAY HAVE, CAN HAVE

In a program we could say CREATE A STUDENT CALLED JOE

(STUDENT is a temporary entity). This is equivalent to reserving space for a

?? 3-dimensional array with pointer called JOE. [That is, "RESERVE JOE(*) AS A".]

\[
\text{STUDENT} \rightarrow \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
3.5 \\
6
\end{bmatrix}
\]

\[
\text{JOE} \rightarrow \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
3.5 \\
6
\end{bmatrix}
\]
SIMSCRIPT data, structures aid in problem definition. Sim model aids in structure & clarity (*self-documenting paper)

3. Each entity is composed of a collection of memory cells called attributes. The attribute values define characteristics of the entity.

4. Definition, organization, & manipulation of entities, attributes, & sets.

2. An entity is a structured data item that represents some element of a modeled system.
TEMPORARY ENTITIES:

Storage is allocated to temporary entities dynamically, as they are created during program execution.

CREATE A[n] entity [CALLED name]
DESTROY entity CALLED name
DESTROY entity "same as ...CALLED entity"

e.g.,
DESTROY THE STUDENT CALLED JOE
DESTROY THIS STUDENT

PERMANENT ENTITIES:

The entire group of n permanent entities of a particular type (e.g., MACHINE) are stored together with the variable N.MACHINE keeping track of the number of entities of that type. E.g.,

PREAMBLE

PERMANENT ENTITIES
EVERY MACHINE HAS AN ID, A DATE, A COST, AND OWNS SOME OPERATORS
END
MAIN

READ N.MACHINE
CREATE EVERY MACHINE
FOR I = 1 TO N.MACHINE,
    READ ID(I), DATE(I), COST(I)

Permanent entities can only be destroyed collectively, as a group:

DESTROY {EVERY, EACH} entity-name
Example:

PRE Amble
NORMally, mode is integer
TEmporary entities
EVERY maN may have a name, an age, a wife, a girlfriend, and an occupation
EVERY woMAn may have a name, an age, a husband, a boyfriend, and an occupation
DEfine name as an alpha variable

END

Main

CREATE A MAN
CREATE A woMAN
LET husband(woMAN) = maN
CREATE A MAN
LET girlfriend(maN) = woMAN
CREATE A woMAN
LET AGE(husband(girlfriend(maN))) = 77.8

STOP

END
**Sets:**

A set is a collection of entities which may be ranked in a particular order (e.g., alphabetically, FIFO, LIFO). An entity may have multiple membership and belong to several sets. Every set must have an owner and possess members (20).

In the Preamble, "EVERY" statements define data structures. For example,

```
EVERY STUDENT HAS A NAME, AN AGE, AND MAY BELONG TO THE CLUB
{DEFINE CLUB AS A FIFO SET} or
{DEFINE CLUB AS A SET RANKED BY LOW NAME} or
{DEFINE CLUB AS A SET RANKED BY LOW NAME THEN BY HIGH AGE}
EVERY CITY OWNS A CLUB
```

There are some predefined variable names used by the SIMSCRIPT compiler and accessible to the programmer: F.SET, the first (ordered) member of SET; L.SET, the last (ordered) member of SET; each member has a predecessor, P.SET, and a successor, S.SET.

In main:

```
CREATE A CITY

CREATE A STUDENT
```
MORE ON SETS:

Every set must have a declared owner and a declared (possible) member. E.g.,

TEMPORARY ENTITIES
EVERY AIRPLANE MAY BELONG TO THE RUNWAY
THE SYSTEM OWNS THE RUNWAY

All these attributes are initialized to zero. Since the set comes into existence with the creation of the owner, the set RUNWAY is "created" as soon as program ("THE SYSTEM") execution begins. At that time, F.RUNWAY, L.RUNWAY, AND N.RUNWAY are all set to zero (null set).

Set membership is not automatic. In order for an entity to belong to a set, it must be "FILE"d in the set. For any entity which may belong to a set, the variable M.SET is equal to 1 if entity is in the set, and zero if it is not. Entities are entered into the set on a first-come-first-served (FIFO) basis unless some other order has been specified.

e.g., CREATE AN AIRPLANE

{FILE AIRPLANE IN RUNWAY} or
{FILE THIS AIRPLANE IN THE RUNWAY}, or
{FILE THIS AIRPLANE LAST IN RUNWAY}
FILE {entity-name} {LAST, FIRST} IN {THE, THIS} set
{arithmetic-expression}

FILE arithmetic-expression {BEFORE, AFTER} arithmetic-exp IN set

REMOVE {THE, THIS} {FIRST, LAST} entity-name FROM set
REMOVE arithmetic-exp FROM set

SPECIAL CONTROL STATEMENTS FOR ENTITIES:

For permanent entities, the statement FOR I = 1 TO N.entity
is the same as FOR EACH entity or FOR EACH entity CALLED
name. Synonyms for EACH are EVERY and ALL.

For temporary or permanent entities, we use the statement:
FOR EACH entity-name OF set. E.g., FOR EVERY JOB IN QUEUE. To
process the entities in a set backwards add the phrase IN REVERSE ORDER.
We may add additional control phrases onto such a FOR statement, e.g.,
(WITH, UNTIL) relational-operator.
Sets provide a higher level of structuring of related data. Sets are organized collections of entities. In data structures terminology, a set would be called a linked list.

For example, every student has a student number, a name, a GPA, a class standing, and may belong to the team, the computer society, and an elite group.

Define team as a FIFO set.
Define computer society as a set ranked by low name.
Define elite group as a set ranked by high GPA.
Every campus owns a team, a computer society, and an elite group.

<table>
<thead>
<tr>
<th>Campus</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>F. Team</td>
<td>Student: No.</td>
</tr>
<tr>
<td></td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td>GPA</td>
</tr>
<tr>
<td>N.</td>
<td>Class Standing</td>
</tr>
<tr>
<td>L.</td>
<td>F. Team</td>
</tr>
<tr>
<td>N.</td>
<td>S.</td>
</tr>
<tr>
<td>F. Computer Society</td>
<td>M.</td>
</tr>
<tr>
<td>L.</td>
<td>P. Computer Society</td>
</tr>
<tr>
<td>N.</td>
<td>S.</td>
</tr>
<tr>
<td>F. Elite Group</td>
<td>M.</td>
</tr>
<tr>
<td>L.</td>
<td>P. Elite Group</td>
</tr>
<tr>
<td>N.</td>
<td>S.</td>
</tr>
</tbody>
</table>

Attributes, pointers indicating relationships to other entities in sets.
<table>
<thead>
<tr>
<th>STUDENT NO</th>
<th>NAME</th>
<th>GPA</th>
<th>CLASS</th>
<th>CLUBs</th>
<th>J.O. or D</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>R.B. PARKER</td>
<td>3.4</td>
<td>UJR</td>
<td>TEAM</td>
<td>JOIN</td>
</tr>
<tr>
<td>20</td>
<td>R. MACDONALD</td>
<td>3.6</td>
<td>LJR</td>
<td>COMP, ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>30</td>
<td>R. CHANDLER</td>
<td>3.2</td>
<td>LSR</td>
<td>TEAM, COMP</td>
<td>JOIN</td>
</tr>
<tr>
<td>40</td>
<td>D. HAMMETT</td>
<td>2.6</td>
<td>USR</td>
<td>TEAM, COMP, ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>50</td>
<td>SPENSER</td>
<td>3.8</td>
<td>LSO</td>
<td>TEAM, ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>50</td>
<td>SPENSER</td>
<td>3.8</td>
<td>LSO</td>
<td>ELITIST</td>
<td>DROP</td>
</tr>
<tr>
<td>40</td>
<td>D. HAMMETT</td>
<td>2.6</td>
<td>USR</td>
<td>ELITIST</td>
<td>DROP</td>
</tr>
<tr>
<td>60</td>
<td>S. SPADE</td>
<td>1.9</td>
<td>UJR</td>
<td>COMP, ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>70</td>
<td>P. MARLOWE</td>
<td>2.2</td>
<td>UJR</td>
<td>TEAM, COMP</td>
<td>JOIN</td>
</tr>
<tr>
<td>80</td>
<td>J.D. MCDONALD</td>
<td>2.5</td>
<td>LJR</td>
<td>TEAM</td>
<td>JOIN</td>
</tr>
<tr>
<td>90</td>
<td>L. ARCHER</td>
<td>3.0</td>
<td>LSR</td>
<td>TEAM</td>
<td>JOIN</td>
</tr>
<tr>
<td>100</td>
<td>T. MAGEE</td>
<td>2.7</td>
<td>UFR</td>
<td>ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>100</td>
<td>T. MAGEE</td>
<td>2.7</td>
<td>UFR</td>
<td>TEAM</td>
<td>JOIN</td>
</tr>
<tr>
<td>100</td>
<td>T. MAGEE</td>
<td>2.7</td>
<td>UFR</td>
<td>ELITIST</td>
<td>DROP</td>
</tr>
<tr>
<td>110</td>
<td>T.M. FLETCHER</td>
<td>2.8</td>
<td>LSO</td>
<td>TEAM, COMP</td>
<td>JOIN</td>
</tr>
<tr>
<td>30</td>
<td>R. CHANDLER</td>
<td>3.2</td>
<td>LSR</td>
<td>TEAM</td>
<td>QUIT</td>
</tr>
<tr>
<td>30</td>
<td>R. CHANDLER</td>
<td>3.2</td>
<td>LSR</td>
<td>ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>10</td>
<td>R.B. PARKER</td>
<td>3.4</td>
<td>UJR</td>
<td>COMP</td>
<td>JOIN</td>
</tr>
<tr>
<td>120</td>
<td>S. HOLMES</td>
<td>4.0</td>
<td>USR</td>
<td>TEAM, ELITIST</td>
<td>JOIN</td>
</tr>
<tr>
<td>130</td>
<td>N. WOLFE</td>
<td>4.0</td>
<td>LSR</td>
<td>COMP, TEAM</td>
<td>JOIN</td>
</tr>
<tr>
<td>140</td>
<td>A. GOODWIN</td>
<td>3.5</td>
<td>UJR</td>
<td>ELITIST</td>
<td>JOIN</td>
</tr>
</tbody>
</table>
THE SOCIAL REGISTER

A growing city has several clubs; new ones are constantly being established. Any person may belong to any number of clubs. We wish to be able to read in new people along with the clubs they wish to join (or leave). If a particular club does not exist when a person wishes to join in, it should be established; if an existing club has no members (after the last member has left), it should be eliminated.

After all applications and resignations have been processed, two sets of alphabetic lists are required:
1) Membership of each club
2) Club membership of each person (Social Register)

Starting with an empty city (no clubs, no people) read in the following data:
(You may add additional people)

<table>
<thead>
<tr>
<th>CLUB</th>
<th>PERSON</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAD</td>
<td>TIMS</td>
<td>JOIN</td>
</tr>
<tr>
<td>KARR</td>
<td>ORSA</td>
<td>JOIN</td>
</tr>
<tr>
<td>ABEL</td>
<td>ACM</td>
<td>JOIN</td>
</tr>
<tr>
<td>FAIN</td>
<td>ACM</td>
<td>JOIN</td>
</tr>
<tr>
<td>FAIN</td>
<td>IEEE</td>
<td>JOIN</td>
</tr>
<tr>
<td>BOND</td>
<td>ACM</td>
<td>JOIN</td>
</tr>
<tr>
<td>FAIN</td>
<td>BRSA</td>
<td>JOIN</td>
</tr>
<tr>
<td>CHAO</td>
<td>ACM</td>
<td>JOIN</td>
</tr>
<tr>
<td>FAIN</td>
<td>IEEE</td>
<td>QUIT</td>
</tr>
<tr>
<td>ABEL</td>
<td>TIMS</td>
<td>JOIN</td>
</tr>
<tr>
<td>BOND</td>
<td>AIIE</td>
<td>JOIN</td>
</tr>
<tr>
<td>KARR</td>
<td>AIIE</td>
<td>JOIN</td>
</tr>
<tr>
<td>CHAO</td>
<td>ACM</td>
<td>QUIT</td>
</tr>
<tr>
<td>KARR</td>
<td>AIIE</td>
<td>JOIN</td>
</tr>
<tr>
<td>FAIN</td>
<td>IEEE</td>
<td>QUIT</td>
</tr>
<tr>
<td>CHAO</td>
<td>TIMS</td>
<td>JOIN</td>
</tr>
</tbody>
</table>

Hints:

PERSON owns a WALLET, belongs to SOCIAL.REGISTER
CARD belongs to WALLET and ROSTER.
CLUB owns a ROSTER, belongs to CLUBS
The SYSTEM owns SOCIAL.REGISTER and CLUBS.

Notes:

1. The program should be able to handle any number of people and clubs.
2. Warning messages should be printed if:
   a) a person tries to join a club in which he/she is already a member
   b) a person tries to leave a club without being a member.
1. // JOB REGION=400K
2. // EXEC SIMSCRIPT,PARM='NOHIST'
3. //SIM.SYSIN DD *
4. PREAMBLE
5. TEMPORARY ENTITIES
6. EVERY CARD HAS A NAME, AN ORG
7. EVERY CLUB OWNS A ROSTER
8. EVERY MEMBER OWNS A SOCIAL.REG
9. EVERY CARD BELONGS TO A ROSTER AND A SOCIAL.REG
10. DEFINE NAME, ORG, T1, T2, J AS TEXT VARIABLES
11. DEFINE ROSTER AS SET RANKED BY LOW ORG, THEN BY LOW NAME
12. DEFINE SOCIAL.REG AS SET RANKED BY LOW NAME, THEN BY LOW ORG
12.5 END
13. MAIN
14. READ J
14.2 CREATE CLUB
14.7 CREATE MEMBER
15. UNTIL J="*"
15.5 DO
18. READ T1, T2
19. FOR EVERY CARD OF ROSTER WITH T1=NAME(CARD) AND T2=ORG(CARD)
20. FIND THE FIRST CASE
21. IF NONE
22. IF J="JO"
23. CREATE CARD
24. LET NAME(CARD)=T1
25. LET ORG(CARD)=T2
26. FILE CARD IN SOCIAL.REG
27. FILE CARD IN ROSTER
28. ELSE IF J="QU"
29. PRINT 1 LINE WITH T1, T2 THUS
30. WARNING: **** IS TRYING TO QUIT **** OF WHICH S/HE IS NOT A MEMBER
31. ENDIF
32. ENDIF
33. ELSE IF J="JO"
34. PRINT 1 LINE WITH T1, T2 THUS
35. WARNING: **** IS TRYING TO JOIN **** OF WHICH S/HE IS ALREADY A MEMBER
36. ELSE IF J="QU"
37. REMOVE CARD FROM ROSTER
38. REMOVE CARD FROM SOCIAL.REG
39. ENDIF
40. ENDIF
40.5 END
READ J
LOOP
PRINT 1 LINE THUS
ROSTERS
FOR EACH CARD IN ROSTER PRINT 1 LINE WITH ORG(CARD), NAME(CARD) THUS

CLUB: **** MEMBER: *****
SKIP 5 OUTPUT LINES
PRINT 1 LINE THUS
SOCIAL REGISTERS
FOR EACH CARD IN SOCIAL.REG PRINT 1 LINE WITH NAME(CARD), ORG(CARD) THUS

MEMBER: **** CLUB: ****
STOP
END
//GO.SYSIN DD *
JO CHAO TIMS
JO KARR ORSA
JO ABEL ACM
JO FAIN ACM
JO FAIN IEEE
JO BOND ACM
JO FAIN ORSA

JO CHAO ACM
QU FAIN IEEE
JO ABEL TIMS
JO BOND AIEEE
JO KARR AIEEE
QU CHAO ACM
JO KARR AIEEE
QU FAIN IEEE
JO CHAO TIMS
*
/*

LEVEL 5 - SIMULATION:

The elements which distinguish a SIMSCRIPT program as a "simulation" rather than just another program are: modeling random phenomena; managing and advancing time; and reporting simulation output statistics.

TIME:

In SIMSCRIPT, time is managed by means of events which may be explicitly declared as event routines or implicit in process routines. When an event occurs, the state of the system is changed in some way. Events occur "instantaneously" in time. Events pre-schedule other events, causing them to occur at some later time ("bootstrapping the system"). A process is a collection of related events occurring over simulated time. The built-in timing routine manages events in simulated time by maintaining a timeclock and calling on appropriate event routines as they are scheduled to occur.
EVENTS:

In an event routine, the word EVENT or UPON replaces the word

ROUTINE:

EVENT ARRIVAL
RETURN
END

The general form for an event routine definition is:

EVENT name (argument list) or EVENT name GIVEN argument list

Event routines are CALLED, not by a CALL statement in the the
user program, but by the timing routine. In the user program we need a

statement of the sort:

SCHEDULE AN ARRIVAL  {AT _____}
{IN _____ (HOURS, MINUTES, DAYS)}
{NOW}

The statement START SIMULATION activates the timing

routine.

An event notice is a temporary entity which represents a single
occurrence of the particular type of event. It has at least five attributes:
TIME.A, the time at which the event is scheduled to occur; EUNIT.A, used for
external events; and P.EV.S, S.EV.S, and M.EV.S, as the event notice is a
member of an event set (EV.S) ordered by TIME.A. Since an event notice is a
temporary entity, it must be declared in the Preamble:

EVENT NOTICES INCLUDE ARRIVAL, WEEKLY.REPORT, AND STOP.SIMULATION
EVERY JOB.OVER HAS A NEXT.JOB AND OWNS SOME PEOPLE
A *process* is a series of events separated by delays: unavoidable delays (e.g., service time) and avoidable delays. A *resource* is a permanent entity which has a capacity, and has two queues assigned to it automatically: Q.RES for demands (customers) waiting and X.RES for demands in service.

To declare a process routine: `PROCESS name (argument list)`

or `PROCESS name GIVEN argument list`. Every occurrence of a process produces a process notice which is filed in the set EV.S along with the event notices. In Preamble, e.g.:

```
PROCESSES INCLUDE MACHINE, JOB, CHECK, JOB
EVERY ALLOCATION, TRAIN HAS A NO. OF. CARS, A DESTINATION
```
THE SIMSCRIPT "WORLD-VIEW"

Process
The description of an object and the sequence of activities which it experiences throughout its "life" in a simulation.

Resource
The description of an object which provides service to the process objects - acting to delay them if already occupied serving another process object.

Entities
Passive objects moved through the system (by processes).

Events
Instantaneous processes.

Sets
Ordered lists of entities, processes, or resources.

Source: CACI
PROCESS "WORLD-VIEW"

A PROCESS MODELS AN OBJECT (STATIC)
AND ITS "LIFE" IN THE SYSTEM (DYNAMIC)

A SIMULATION MAY CONTAIN:
MANY COPIES OF A SINGLE PROCESS
MANY DIFFERENT PROCESSES

PROCESSES INTERACT BY:
CHANGING SYSTEM STATE
DIRECT COMMUNICATION
COMPETING FOR RESOURCES

SIMPLE PROCESS COMMANDS
ACTIVATE A JOB NOW
ACTIVATE A SHIFT-CHANGE IN 8 HOURS
WAIT X DAYS
WORK 8 HOURS

SOURCE: CACI
REPRESENTING SIMULATED TIME

TIME UNITS

DAYS (UNITS)
HOURS (HOURS>V = 24)
MINUTES (MINUTES>V = 60)

USED IN PROCESS SCHEDULING, SUCH AS
ACTIVATE A CUSTOMER IN 5 MINUTES

OR, WITHIN A PROCESS,
WAIT 15 MINUTES

OR,

WORK 3 DAYS

(WORK AND WAIT ARE INTERCHANGEABLE)

INTERNALLY, TIME IS RECORDED IN DAYS
TIME>V IS THE SYSTEM'S "CLOCK"

Source: CACI

31
PROGRAM STRUCTURE

PREAMBLE
DEFINES ALL MODEL COMPONENTS STATICALLY
(PROCESSES, RESOURCES, ENTITIES, EVENTS, SETS)
DEFINES ALL GLOBAL VARIABLES
DEFINES ALL STATISTICS TO BE GATHERED

END
MAIN
EXECUTION STARTS HERE
::
START SIMULATION
(PASS CONTROL TO TIMING ROUTINE)
::
END
(TIMING ROUTINE)
AUTOMATICALLY GENERATED FROM PREAMBLE
SEQUENCES EXECUTION OF PROCESS AND EVENTS

PROCESS NAME1
::
END
PROCESS NAME2
::
END
Etc

Source: CAC
ENTITIES

ENTITIES INCLUDE

TEMPORARY ENTITIES
PROCESSES
PERMANENT ENTITIES
RESOURCES

ANY ENTITY MAY

HAVE ATTRIBUTES
BELONG TO A SET (OF ENTITIES)
OWN A SET (OF ENTITIES)

SOURCE: CACI
NAMES OF ENTITIES

EVERY ENTITY HAS A "NAME" AND A VARIABLE WITH THE SAME NAME

THE VARIABLE "POINTS" TO THE ENTITY

IN SOME INSTANCES, THE GENERIC NAME MUST BE USED

IN OTHER INSTANCES, ANY VARIABLE MAY BE USED TO REFERENCE AN ENTITY

Source: CACI
MORE ON PROCESSES

Program Structure

PREAMBLE

all processes, resources, events, etc. defined

END

MAIN

program execution begins here.
Resources created and initialized.
Initial process objects activated. (So timing states have something to pick up when simulation begins.)

END

PROCESS name

END
Processes

A process represents an object and the sequence of actions it experiences throughout its life in the model.

There may be many copies (instances) of a process.

There may be many different processes in a model.

A process may be thought of as a sequence of interrelated events separated by lapses of time (either predetermined or indefinite). 
Examples: service time, competition for resources.

When a process is active, it executes the statements contained in the process routine.
THE PENDING LIST

EACH PROCESS ACTIVATION CREATES A NEW PROCESS "NOTICE." THIS IS A BLOCK OF DATA WHICH DESCRIBES THE PROCESS INSTANCE AS IT PROCEEDS.

THERE MAY BE MANY PROCESS NOTICES IN THE SYSTEM AT THE SAME TIME. ALL BUT ONE OF THEM ARE FOUND IN THE "PENDING LIST."

IN THE FOLLOWING DIAGRAM, EACH BOX REPRESENTS A PROCESS NOTICE WITH THE FOLLOWING INFORMATION:

- PROCESS NAME AND INSTANCE
- NEXT "MOVE" TIME
- NEXT LINE TO BE EXECUTED
- SYSTEM INFORMATION
START SIMULATION

- ANY PROCESSES ON PENDING LIST?
  - YES
    - SELECT PROCESS WITH EARLIEST (RE)ACTIVATION TIME
    - UPDATE CLOCK TO TIME OF EVENT
    - DETERMINE TYPE OF PROCESS
    - REMOVE PROCESS FROM PENDING LIST
    - EXECUTE PROCESS ROUTINE
  - NO
    - RETURN

Termination of Timing routine may also occur due to STOP statement.

Basic SIMSCRIPT II.5 Timing Routine
RESOURCE ALLOCATION

1) DECLARE NAMES OF RESOURCES IN PREAMBLE

2) "CREATE" RESOURCES BEFORE ATTEMPTING TO USE

N RESOURCE - A GLOBAL VARIABLE CONTAINING NUMBER
OF "KINDS" OF RESOURCE

U RESOURCE(I) - A GLOBAL VARIABLE CONTAINING NUMBER
OF "UNITS" OF RESOURCE (I) CURRENTLY AVAILABLE
REQUEST/RELINQUISH AFFECT U RESOURCE(I)

N.X RESOURCE(I) - A GLOBAL VARIABLE CONTAINING NUMBER
OF REQUESTS FOR RESOURCE(I) CURRENTLY BEING FILLED

N.Q RESOURCE(I) - A GLOBAL VARIABLE CONTAINING NUMBER
OF REQUESTS FOR RESOURCE(I) CURRENTLY OUTSTANDING
(I.E., NOT BEING FILLED)
A SUMMARY OF RESOURCE MODELING

1) THE NUMBER OF SUBGROUPS IS ESTABLISHED AT THE BEGINNING OF EXECUTION (N RESOURCE)
2) EACH SUBGROUP HAS ITS OWN RESOURCE ALLOCATION (U RESOURCE(I)) AND QUEUE
3) RESOURCE SUBGROUPS MAY HAVE ATTRIBUTES. ALL UNITS WITHIN A SUBGROUP ARE CONSIDERED IDENTICAL
4) SELECTION OF A SUBGROUP IS MADE BEFORE REQUEST
5) REQUEST IS A COMMITMENT TO WAIT UNTIL SERVED
6) A PRIORITY SCHEME IS AVAILABLE
   REQUEST 1 PUMP(I) WITH PRIORITY N
   (WHERE N IS AN INTEGER CONSTANT, VARIABLE, OR EXPRESSION)
   A HIGH VALUE OF N CAUSES THE REQUEST TO BE PLACED NEARER THE FRONT OF THE QUEUE
7) THE PRIORITY DOES NOT CAUSE PRE-EMPTION OF ALLOCATED RESOURCES
Resources      passive entities

A resource is a object required by a process. If the resource is not available,
the process object is placed in a queue.

N. resource = # of "kinds" of resource, each with its own queue
N. Q. resource = # in queue
U. resource = # identical units of resource currently available
N. X. resource = # process objects being executed

N. TELLER = 1
N. Q. TELLER = 5
U. TELLER = 4
N. X. TELLER = 4

N. TELLER = 4
N. Q. TELLER (1) = 3  U. TELLER (1) = 1
N. Q. TELLER (2) = 2  U. TELLER (2) = 1
N. Q. TELLER (3) = 1  U. TELLER (3) = 1
N. Q. TELLER (4) = 4  U. TELLER (4) = 1
N. X. TELLER (1) = 1

Or, you can name 4 different resources, e.g., TELLER1, TELLER2, etc.
Resources must be created and initialized before they can be used by processes.

A resource acts to delay a process object when it is already engaged by other process objects. It automatically "releases" the process object when allocation can be made.

REQUEST 1 PUMP(1)
REQUEST 1 WORKERS
RELINQUISH 1 RUNWAY.
Consider a small gas station where customers arrive randomly, queue up for service, receive service, and leave. Our goal might be to determine the effects of adding or deleting pumps (or attendants) from the system. For the moment, however, our real goal is merely to construct and execute a very simple SIMSCRIPT model.

To introduce randomness, assume that we have a source of uniformly distributed random numbers for which we can establish bounds. In SIMSCRIPT this is referenced as UNIFORM.F. This function has three parameters: the lower bound, the upper bound, and a random number stream. Each time the function is executed, a new sample from the interval is computed.

```
1 /// EXAMPLE 1 A SIMPLE GAS STATION MODEL
2
3 PREAMBLE
4 PROCESSES INCLUDE GENERATOR AND CUSTOMER
5 RESOURCES INCLUDE ATTENDANT
6 END
7
8 MAIN
9 CREATE EVERY ATTENDANT(1)
10 LET G.ATTENDANT(1) = 2
11 ACTIVATE A GENERATOR NOW
12 START SIMULATION
13 END
14
15 PROCESS GENERATOR
16 FOR I = 1 TO 1000,
17 DO
18   ACTIVATE A CUSTOMER NOW
19   WAIT UNIFORM.F(2.0,8.0,1) MINUTES
20 LOOP
21 END
22
23 PROCESS CUSTOMER
24 REQUEST 1 ATTENDANT(1)
25 WORK UNIFORM.F(5.0,15.0,2) MINUTES
26 RELINQUISH 1 ATTENDANT(1)
27 END
```
<table>
<thead>
<tr>
<th>TIME</th>
<th>CURRENT PROCESS</th>
<th>AVAILABLE ATTENDANTS</th>
<th>CUSTOMERS IN QUEUE</th>
<th>PENDING PROCESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>[AT START SIMULATION]</td>
<td>2</td>
<td>NONE</td>
<td>[G₁,1,0.0]</td>
</tr>
<tr>
<td>0.0</td>
<td>[G₁,1,0.0]</td>
<td>2</td>
<td>NONE</td>
<td>[C₁,1,0.0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,7.547]</td>
</tr>
<tr>
<td>0.0</td>
<td>[C₁,1,0.0]</td>
<td>1</td>
<td>NONE</td>
<td>[C₁,4,6.847]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,7.547]</td>
</tr>
<tr>
<td>6.847</td>
<td>[C₁,4,6.847]</td>
<td>2</td>
<td>NONE</td>
<td>[G₁,6,7.547]</td>
</tr>
<tr>
<td>7.547</td>
<td>[G₁,6,7.547]</td>
<td>2</td>
<td>NONE</td>
<td>[C₂,1,7.547]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,10.617]</td>
</tr>
<tr>
<td>7.547</td>
<td>[C₂,1,7.547]</td>
<td>1</td>
<td>NONE</td>
<td>[C₁,6,10.617]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td>10.617</td>
<td>[G₁,6,10.617]</td>
<td>1</td>
<td>NONE</td>
<td>[C₃,1,10.617]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,14.073]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
<tr>
<td>14.073</td>
<td>[G₁,6,14.073]</td>
<td>0</td>
<td>NONE</td>
<td>[C₄,1,14.073]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,21.932]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
<tr>
<td>14.073</td>
<td>[C₄,1,14.073]</td>
<td>0</td>
<td>none</td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,21.932]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
<tr>
<td>14.400</td>
<td>[C₂,4,14.400]</td>
<td>0</td>
<td>none</td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,21.932]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
<tr>
<td>14.400</td>
<td>[C₄,3,14.400]</td>
<td>0</td>
<td>none</td>
<td>[C₂,4,14.400]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[G₁,6,21.932]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[C₃,4,24.367]</td>
</tr>
</tbody>
</table>

Detailed Trace of Execution of Example 1
PREAMBLE

PROCESSSES INCLUDE GENERATOR AND CUSTOMER
RESOURCES INCLUDE ATTENDANT
ACCUMULATE AVG.QUEUE.LENGTH AS THE AVERAGE
AND MAX.QUEUE.LENGTH AS THE MAXIMUM
OF N.Q.ATTENDANT
ACCUMULATE UTILIZATION AS THE AVERAGE OF N.X.ATTENDANT

MAIN
CREATE EVERY ATTENDANT(1)
LET U.ATTENDANT(1) = 2
ACTIVATE A GENERATOR NOW
START SIMULATION
PRINT 4 LINES WITH AVG.QUEUE.LENGTH(1), MAX.QUEUE.LENGTH(1),
AND UTILIZATION(1) * 100. / 2

SIMPLE GAS STATION MODEL WITH 2 ATTENDANTS
AVERAGE CUSTOMER QUEUE LENGTH IS **
MAXIMUM CUSTOMER QUEUE LENGTH IS **
THE ATTENDANTS WERE BUSY ** PER CENT OF THE TIME.

PROCESS GENERATOR
FOR I = 1 TO 1000,
DO
ACTIVATE A CUSTOMER NOW
WAIT UNIFORM.F(2.0,8.0,I) MINUTES
LOOP
END

PROCESS CUSTOMER
REQUEST 1 ATTENDANT(1)
WORK UNIFORM.F(5.0,15.0,2) MINUTES
RELINQUISH 1 ATTENDANT(1)
END

SIMPLE GAS STATION MODEL WITH 2 ATTENDANTS
AVERAGE CUSTOMER QUEUE LENGTH IS 7.809
MAXIMUM CUSTOMER QUEUE LENGTH IS 21
THE ATTENDANTS WERE BUSY 98.64 PER CENT OF THE TIME.

Output from Example 1A
To illustrate the ease of doing this in SIMSCRIPT, let us modify our example as follows:

a. Customers choose the type of gas they require (premium or regular) and must wait for the proper pump to become available. Let us arbitrarily assume that 70% require premium and 30% require regular. Initially, we shall arbitrarily have one regular pump and three premium pumps.

b. An attendant is not required until the pump is available. The attendant starts the pump and then is free until the pump stops. He is then required in order to complete the service.

c. We wish to measure the queues and utilization for the attendants and two types of pumps.
EXAMPLE 18 A MORE ELABORATE GAS STATION MODEL

PREAMBLE

PROCESSES INCLUDE GENERATOR AND CUSTOMER
RESOURCES INCLUDE ATTENDANT AND PUMP
ACCUMULATE AVG.ATTENDANT.QUEUE.LENGTH AS THE AVERAGE
AND MAX.ATTENDANT.QUEUE.LENGTH AS THE MAXIMUM
OF N.Q.ATTENDANT
ACCUMULATE UTILIZATION AS THE AVERAGE OF N.X.ATTENDANT
ACCUMULATE AVG.PUMP.QUEUE.LENGTH AS THE AVERAGE
AND MAX.PUMP.QUEUE.LENGTH AS THE MAXIMUM
OF N.Q.PUMP
ACCUMULATE PUMP.UTILIZATION AS THE AVERAGE OF N.X.PUMP
DEFINE .REGULAR TO MEAN 1
DEFINE .PREMIUM TO MEAN 2

DEFINE GRADE AS AN INTEGER VARIABLE
IF RANDOM.F(3) > 0.70,
LET GRADE = .REGULAR
ELSE
LET GRADE = .PREMIUM

ALWAYS REQUEST 1 PUMP(GRADE)
REQUEST 1 ATTENDANT(1)
WORK UNIFORM.F(2.0,4.0,2) MINUTES
RELINQUISH 1 ATTENDANT(1)
WORK UNIFORM.F(5.0,9.0,2) MINUTES
REQUEST 1 ATTENDANT(1)
WORK UNIFORM.F(3.0,5.0,2) MINUTES
RELINQUISH 1 ATTENDANT(1)
RELINQUISH 1 PUMP(GRADE)

SIMPLE GAS STATION WITH TWO ATTENDANTS
AND TWO GRADES OF GASOLINE
AVERAGE QUEUE WAITING FOR ATTENDANTS IS *•••• CUSTOMERS
MAXIMUM " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " "
SIMPLE GAS STATION WITH TWO ATTENDANTS
AND TWO GRADES OF GASOLINE
AVERAGE QUEUE WAITING FOR ATTENDANTS IS .138 CUSTOMERS
MAXIMUM = 2
THE ATTENDANTS WERE BUSY 69.93 PER CENT OF THE TIME.

THE QUEUES FOR THE PUMPS WERE AS FOLLOWS:
GRADE | AVERAGE | MAXIMUM | UTILIZATION
REGULAR: 1.237 | 8 | 79.94 PERCENT
PREMIUM: .084 | 3 | 70.74 PERCENT
Output Simulation Statistics

TALLY (Time-independent Statistics)

Accumulate (Time-dependent Statistics)

TALLY NO. OF TOWS AS THE NUMBER,
AVG. TOW. DELAY AS THE AVERAGE,
MAX. MAXIMUM OF DELAY TIME.

Accumulate AVG. QUEU LENGTH AS THE AVERAGE OF N. QUEUE

DIRECTION

DELAY TIME, MAX, MIN AT, MAXSAT, P. QUEUE, L. QUEUE, M. QUEUE

TOW

TOWARRTIME

P. QUEUE (4)
S. QUEUE (4)
M. QUEUE (4)
P.Q. (2)
S.Q. (2)
M.Q. (2)
Figure 4.14 - Reset Example
MORE ELABORATE PERFORMANCE MEASURING

QUALIFYING STATISTICS

SELECTIVE resetting

EXAMPLE:

ACCUMULATE WK.MEAN AS THE WEEKLY MEAN
AND MON.MEAN AS THE MONTHLY MEAN
AND GRAND.AVG AS THE MEAN OF N.Q. TELLER

PROCESS WEEKLY REPORT

UNTIL TIME.V > .END.OF.SIMULATION,
DO
  WAIT 7 DAYS
  PRINT 1 LINE WITH TIME.V AND WK.MEAN (1) THUS
  AT DAY * AVG = **
  RESET WEEKLY TOTALS OF N.Q.TELLER (1)
LOOP

END
### TABLE 4.1

ACCUMULATE/TALLY STATISTICS

<table>
<thead>
<tr>
<th>STATISTIC</th>
<th>ACCUMULATE COMPUTATION</th>
<th>TALLY COMPUTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER</td>
<td>$K$ — THE NO. OF CHANGES IN $X$</td>
<td>$N$ — THE NO. OF SAMPLES OF $X$</td>
</tr>
<tr>
<td>SUM</td>
<td>$\sum (X(TIME.V - T_L))$</td>
<td>$\sum X$</td>
</tr>
<tr>
<td>MEAN</td>
<td>$\frac{\text{SUM}}{(TIME.V - T_0)}$</td>
<td>$\frac{\text{SUM}}{N}$</td>
</tr>
<tr>
<td>SUM.OF.SQUARES</td>
<td>$\sum (X^2(TIME.V - T_L))$</td>
<td>$\sum X^2$</td>
</tr>
<tr>
<td>MEAN.SQUARE</td>
<td>$\frac{\text{SUM.OF.SQUARES}}{(TIME.V - T_0)}$</td>
<td>$(\sum X^2)$</td>
</tr>
<tr>
<td>VARIANCE</td>
<td>MEAN.SQUARE$^\text{2} - \text{MEAN}^\text{2}$</td>
<td>MEAN.SQUARE$^\text{2} - \text{MEAN}^\text{2}$</td>
</tr>
<tr>
<td>STD.DEV</td>
<td>$\sqrt{\text{VARIANCE}}$</td>
<td>$\sqrt{\text{VARIANCE}}$</td>
</tr>
<tr>
<td>MAXIMUM</td>
<td>$M = \text{MAXIMUM} (X)$ FOR ALL $X$</td>
<td>$M = \text{MAXIMUM} (X)$ FOR ALL $X$</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>$m = \text{MINIMUM} (X)$ FOR ALL $X$</td>
<td>$m = \text{MINIMUM} (X)$ FOR ALL $X$</td>
</tr>
</tbody>
</table>

NOTES:

$TIME.V$ = current simulated time  
$T_L$ = simulated time at which variable was set to its current value  
$T_0$ = simulated time of last reset for this variable  
$x$ = sample value of variable before change occurs

103