AD-A065 352 DEFENSE COMMUNICATIONS ENGINEERING CENTER RESTON VA F/G 9/2
SOL-370 LANGUAGE REFERENCE MANUAL.(U)
SEP 78 H ULFERS

UNCLASSIFIED DCEC-TN-11-78 SBIE-AD-E100 172 NL

1 OF 1
AD
A065352

AD-E100 172

(2) LEVEL III

AD A065352

DDC FILE COPY

TN 11-78

DEFENSE COMMUNICATIONS ENGINEERING CENTER

TECHNICAL NOTE NO. 11-78

SOL-370 LANGUAGE REFERENCE MANUAL

SEPTEMBER 1978

DDC
RECEIVED
MAR 7 1979
B

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

79 02 16 019

UNCLASSIFIED June 1978

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DCEC-TN-11-78
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): SOL-370 Language Reference Manual.
5. TYPE OF REPORT & PERIOD COVERED: Technical Note
6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): H. Ulfers — Horst Ulfers
8. CONTRACT OR GRANT NUMBER(s): 35p.
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Defense Communications Engineering Center, Computer Systems Division, R800, 1860 Wiehle Ave., Reston, VA 22090
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: N/A
11. CONTROLLING OFFICE NAME AND ADDRESS: Same as 9
12. REPORT DATE: Sep 1978
13. NUMBER OF PAGES: 34
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): N/A — SBIE — AD-E100 172
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A
16. DISTRIBUTION STATEMENT (of this Report)

A. Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

N/A

18. SUPPLEMENTARY NOTES

Review relevance 5 years from submission date.

DDC RECEIVED MAR 7 1979 B

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

SOL-370
SOL Model
Arithmetic Language
Syntax Description
Simulation
Language

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document describes the SOL-370 algorithmic language, which is used to construct general systems models for simulation. After the language syntax is described, several sample models are given. This document replaces DCEC TN 25-75, on the same subject.

A025 927

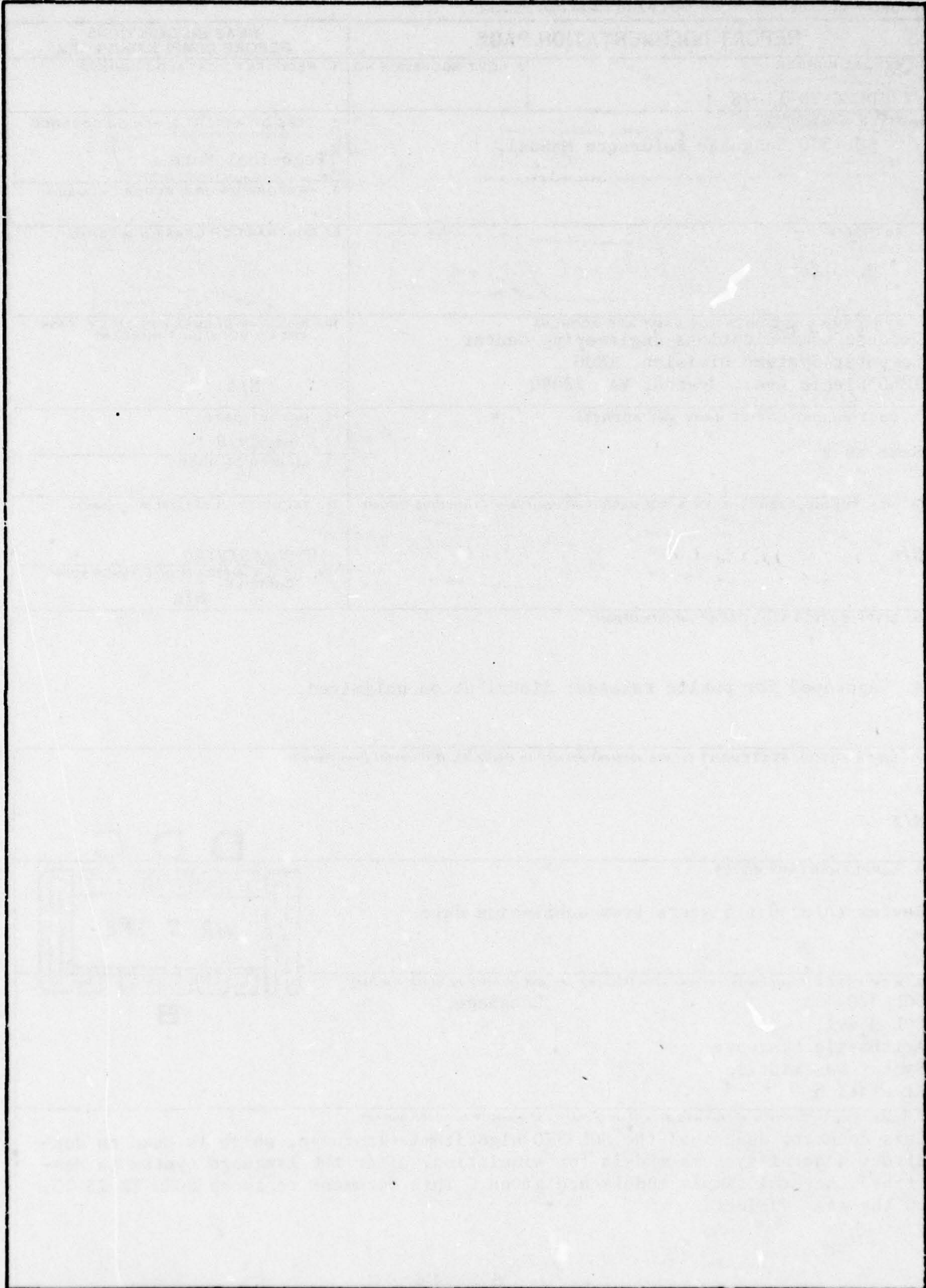
79 02 16 019

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

407 519

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



TECHNICAL NOTE NO. 11-78

SOL-370

LANGUAGE REFERENCE MANUAL

Sept 1978

Prepared by:

Horst Ulfers

Approved for Publication:

ROBERT E. LYONS
Chief, Computer Systems Division

FOREWORD

The Defense Communications Engineering Center (DCEC) Technical Notes (TN's) are published to inform interested members of the defense community regarding technical activities of the Center, completed and in progress. They are intended to stimulate thinking and encourage information exchange; but they do not represent an approved position or policy of DCEC, and should not be used as authoritative guidance for related planning and/or further action.

Comments or technical inquiries concerning this document are welcome, and should be directed to:

Director
Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, Virginia, 22090

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
II.	SOL SYNTAX	2
	1. The Syntax Notation	2
	2. The Model Structure	4
	3. Identifiers and Constants	5
	4. Expressions and Relations	6
	5. Facilities and Associated Commands	8
	6. Stores and Associated Commands	10
	7. Trunks and Associated Commands	11
	8. Transactions and Associated Commands	13
	9. Special SOL Statements	15
	10. Compound and Conditional Statements	17
III.	SAMPLE MODELS	18
	BIBLIOGRAPHY	30

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

I. INTRODUCTION

This manual replaces TN 25-75, SOL-370 Language Reference Manual and Users' Guide. The syntax description in this document reflects the latest updates as implemented in SOL-370 Rel. 6/78.

SOL-370 is a dialect of the SOL-simulation language as developed by Knuth and McNeley* . The original SOL language has been extended to accomodate algorithms essential for the simulation of communications networks and systems. Furthermore, SOL-370 has been implemented as an extension of the PL/I language. This allows for a free intermixing of SOL-370 and PL/I language statements.

SOL-370 is an algorithmic language used to construct models of general systems for simulation in a readable form. The model builder describes his model in terms of PROCESSES whose number and detail are completely arbitrary and definable within the constraints of the language elements. A SOL model consists of a number of statements and declarations which have a character similar to that found in programming languages such as PL/I and ALGOL.

The model is not built to be executed in a sequential fashion, as ordinary programming languages require. Rather, the processes are written and executed as though all were running in parallel. Control among processes is maintained by the interaction of GLOBAL ENTITIES and by control and communications instructions within the different processes. At the initiation of the simulation all processes are begun simultaneously.

Variables declared within a process are called LOCAL VARIABLES. Within a given process it is possible to have several actions occurring at once; therefore, to visualize the process, we may think of several objects on which the action takes place, each in its own place in the process at any given time. These objects will be referred to as TRANSACTIONS. A set of local variables corresponding in number to those declared in the process is "carried with" each transaction of that process. Transactions situated within one process may not refer to the local variables of another process nor to the local variables of another transaction in the same process.

GLOBAL ENTITIES are of four major types: GLOBAL VARIABLES, FACILITIES, STORES, and TRUNKS. Global variables can be referenced or changed by any transaction from any process in the system, and the variable possesses only one value at any given time.

---

* D. E. Knuth and J. L. McNeley, "A Formal Definition of SOL," IEEE Transactions on Electronic Computers, EC-13 No. 5 (Aug 1964) pp 409-414

II. SOL SYNTAX

1. THE SYNTAX NOTATION

The Backus Normal Form (BNF) is used to describe the syntax of the SOL language. The following rules explain the use of this notation.

a. The Notation Variable. A Notation Variable is the name for a class of elements used in a programming language. It consists of letters and hyphens and is enclosed in "less than" and "greater than" symbols.

EXAMPLES:

<digit> This denotes the occurrence of a digit, which may assume a value within the range of 0 through 9.

<facility name> This denotes the occurrence of a Notation Variable of the class Facility Name.

<do statement> This denotes the occurrence of a DO statement.

b. The Notation Constant. A Notation Constant is the literal occurance of a string of characters. It is represented in capital letters.

Example:

STORE This denotes the literal occurrence of the word "STORE".

c. The Syntactical Unit. A Syntactical Unit is defined as a single variable, a constant, or any collection of notation variables, notation constants, and syntax language symbols. The vertical stroke "|" separating two Syntactical Units indicates a choice, which can be made between the two. Anything enclosed in brackets denotes an option. The syntax within the brackets may be used or left out.

EXAMPLES:

<identifier> FIXED|FLOAT This denotes an identifier which may have the attribute FIXED or FLOAT.

<identifier>[(<constant>)] This denotes an identifier which may optionally be subscripted by a number.

d. <u>General Format.</u> The syntax statement is composed of a notation variable separated from the following syntactical unit by the definition symbol "::=" .

EXAMPLE:

<go to statement> ::= GO TO <label>;

2. THE MODEL STRUCTURE

When coding a SOL model the format below should be followed:

```
<global variable declarations> ]
<resource declarations>         } GLOBAL DECLARATIONS  ]
<table declarations>           ]                       |
                                                       |
<process declaration>          ]                       |
<process statements>            } FIRST PROCESS        |
<end statement>                ]                        } MODEL
                                                       |
<process declaration>          ]                       |
<process statements>            } SECOND PROCESS       |
<end statement>                ]                       |
  .  .  .  .                                           |
  .  .  .  .                                           |
<end statement>                                        ]
```

First, all global declarations should be listed. The order of the global declarations is arbitrary. The order in which processes are listed should be selected carefully, since the first process will be started first and if any values are read for global variable initialization, this should be done in the first process. The program should be followed by an END statement. If not included, the system will provide this statement.

a. Process Declaration

```
<process description>::=PROCESS <identifier>
                        [,T=<constant>] [,R=<constant>];
```

Each process is bracketed with the process declaration and an END statement. The two optional parameters allow the modeler to specify the maximum number of simultaneous transactions T and the maximum number of resources R (facilities, stores, and trunks) encountered by any one transaction. This feature is important in optimizing the model's core requirements.

EXAMPLE:

```
PROCESS SWITCH, T=100, R=4;
```

b. <u>Variable Declaration.</u> Variables are declared either at the beginning of the program outside the processes as GLOBAL VARIABLES, or at the beginning of each process as LOCAL VARIABLES.

```
<global variable declaration>:=<variable declaration>

<variable declaration>:=INTEGER <identifier list> ;
                       | REAL <identifier list>;
```

When declared INTEGER, their internal representation will be fixed binary; when declared REAL, floating decimal.

EXAMPLES:

```
INTEGER A, B, C, D;
REAL X, Y, Z;
INTEGER AR(16);
INTEGER BAR(0:100);
```

c. <u>Resource Declarations</u>

```
<resource declaration> ::= <facility declaration>;
                         | <store declaration>;
                         | <trunk declaration>;
```

All resources must be declared ahead of the process declarations. For details see the discussion for the specific resource.

3. IDENTIFIERS AND CONSTANTS

```
<letter>::=A|B|C|D|...|Z
<digit>::=0|1|2|3|...|9
<constant>::=<constant>|<decimal constant>
<constant>::=<digit>
<time>::= - | +
<decimal constant>::=<constant>.<constant>[E<sign><constant>]
<identifier>::=<letter>|<identifier><letter>|
               <letter><digit>
```

Specific identifiers are used as the names of global variables, resources, statistical tables, processes, and procedures. A specific identifier can be used for only one purpose in a program. Constants are used to represent integer numbers; decimal constants represent real numbers. Identifiers must be declared before they are used elsewhere. All SOL commands and variables starting with '$' are reserved words and should not be used as identifiers.

4. EXPRESSIONS AND RELATIONS

```
<name>::=<identifier>|<identifier>(<expression>)
```

By variable name, facility name, etc., we will mean that the identifier in the name has appeared in a variable declaration, facility declaration, etc., respectively.

```
<primary>::=<variable name>|<store name>|
                <constant>|<decimal constant>|TIME|
                (<expression>)|ABS(<expression>)|
                DISTRIBUTION((a)x,(b)y,...(c)z)|
                NORMAL(<expression>,<expression>)|
                EXPONENTIAL(<expression>)|POISSON(<expression>)|
                GEOMETRIC(<expression>)|RANDOM
<term>::=<primary>|<term><primary>|<term>,<primary>|
            MOD(<term>,<primary>)
<sum>::=<term>|+<term>|-<term>|<sum>+<term>|<sum>-<term>
<unconditional expression>::=<sum>|<digit>:<digit>
<expression>::=<unconditional expression> |
               if <relation> THEN <expression> ELSE <expression>
```

The meaning of an arithmetical operation inside an expression is identical to the meaning in PL/1.

The new elements here are "MOD(a,b)," the positive remainder obtained upon dividing a by b ; "MAX(a,b,c,.....)" and "MIN(a,b,c,...)," which denote the maximum and minimum values, respectively, of the expressions in parentheses; and there are also notations for expressing random values. The expression DISTRIBUTION ((a)x,(b)y,...(c)z)) indicates a random selection between integers x, y, and z with the respective weights a,b, and c.

EXAMPLE :

```
A=DISTRIBUTION((4)1,(2)2,(3)4,(9)9);
```

The expressions NORMAL(M,S), POISSON(M), GEOMETRIC(M), and EXPONENTIAL(M) indicate random values with special distributions which occur frequently in applications. A random number drawn from the normal distribution with mean M and standard deviation S is denoted by NORMAL(M,S) and is a real (not necessarily integer) value. A number drawn from the exponential distribution with mean M is denoted by EXPONENTIAL(M) and is also of type real. The poisson distribution signified by POISSON (M), on the other hand, yields only integer values; e.g. the probability that POISSON(M) = n is $(e^{-M}M^{n}/n!)$. The geometric distribution with mean M, denoted by GEOMETRIC(M), also yields integer

values, where the probability that GEOMETRIC(M)=N-1 is 1/M(1-1/M). The symbol RANDOM denotes a random real number between 0 and 1 having a uniform distribution. Finally, the notation a:b denotes a random integer between the limits a and b. The normal, exponential, poisson, and geometric distributions are mathematically expressible in terms of random distributions as follows:

$$NORMAL(M,S) = S * \sqrt{-2 \ln(RANDOM)} * \sin(2\pi * RANDOM) + M$$

$$EXPONENTIAL(M) = - M \ln(RANDOM)$$

$$POISSON(M) = n \text{ if } e^{-M}(1+M+M^2/2!+.....+M^{n-1}/(n-1)!) <= RANDOM < e^{-M}(1+M + ... + M^n/n!)$$

$$GEOMETRIC(M) = (1 + \ln(RANDOM)/\ln(1-1/M)).$$

As examples of the use of these distributions, consider a population of customers coming to a market with an average of one customer every M minutes. The distribution of waiting time between successive arrivals is EXPONENTIAL(M). On the other hand, if an average of M customers come in per hour, the distribution of the actual number of customers arriving in a given hour is POISSON(M). If an individual performs an experiment repeatedly with a chance of success, I/M, on each independent trial, the number of trials needed until he first succeeds is GEOMETRIC(M).

The special symbol "TIME" indicates the current time; initially, time is zero. The value of a store name is the capacity remaining in the store.

```
<relational operator> ::= = | ¬= | <= | >= | > | <
<relation primary>::= <unconditional expression>
              <relational operator> <unconditional expression>|
              <facility name> BUSY |<facility name> NOT BUSY |
              <store name> FULL | <store name> NOT FULL |
              <store name> EMPTY | <store name> NOT EMPTY |
              PROBABILITY <expression> | (<relation>)
<relation> ::= <relation primary>|
              <relation primary> | <relation primary>|
              <relation primary> & <relation primary>|
              ¬<relation primary>
```

These relations have obvious meanings except for the construction "PROBABILITY e ," which stands for a random condition that is true with probability e. (Here e must be less than or equal to 1.)

IF PROBABILITY 0.12 THEN(12% of the time) ELSE(88% of the time).

5. FACILITIES AND ASSOCIATED COMMANDS

A FACILITY is a global element which can be controlled by only one transaction at a time. Associated with each request for the facility is a "control strength," and if a requesting transaction has a higher strength than the transaction controlling the facility, an interrupt will occur. Interrupts may be nested to any depth. If the requesting transaction is not of greater strength than the controlling transaction, then the requesting transaction stops and waits for the facility until the controlling transaction releases its control.

The following declarations and commands are associated with facilities.

a. Facility Declaration

```
<facility declaration> ::= FACILITY <facility name list>;
<facility name list>::=<facility name>[,<facility name list>]
<facility name>::= <name>[(<constant>)]
```

Facilities are declared at the beginning of the program ahead of the processes. Facilities may be declared as one-dimensional arrays.

```
EXAMPLES:      FACILITY TERMINAL;
               FACILITY LINE (16);
```

b. SEIZE Statements

```
<seize statement>::= SEIZE <facility name>;|
                     SEIZE <facility name>, <expression>;
```

The first form is equivalent to "SEIZE <facility name>, 0." This statement is usually rather simple, but there are situations when complications arise. If the facility is not busy when this statement occurs, then it becomes busy at this point and remains busy until later released by this transaction. (Note: If this transaction creates another transaction, the new transaction does not control the facility.) The <expression> in the SEIZE statement represents the "control strength" which is normally zero. Allowance is made, however, for one transaction to interrupt another. For example, if the facility is busy when the seize statement occurs, let CS be the control strength with which the facility was seized and let HS be the control strength of this seize statement. If HS <= CS , the transaction executing the SEIZE statement waits until the facility is not busy. If HS > CS , however, interrupt occurs. The preempted transaction is handled according to the last INTERRUPT statement it executed. The transaction, A, which had control of the facility, is stopped wherever it was in its process, and the present transaction, B, seizes the facility. When B releases the facility, the following occurs:

(1) If A was executing a wait statement when interrupted, the time of wait is increased by the time which passed during the interrupt.

(2) There may be several transactions waiting but not attempting to seize this facility. If any of these has a higher control strength than CS, then A is interrupted again. The transaction which interrupts is chosen by the normal rules for deciding who obtains control of a facility upon release, as described in the section for the RELEASE STATEMENT.

The control strength in the present implementation of SOL must be an integer between 0 and 15. This allows interrupts to be nested up to 15 deep.

EXAMPLES:

```
SEIZE TERMINAL, PRIORITY_CLASS;
SEIZE LINE, 10;
SEIZE PUMP;
SEIZE TERMINAL, 1:10;
SEIZE LINE, EXPONENTIAL(15);
SEIZE LINE, A*(B-C);
```

c. RELEASE Statement

<release statement>::=RELEASE <facility name>;

This statement is permitted only when the transaction is actually controlling the facility because of a previous seizure. When the facility is released, there may be several other transactions waiting because of seize statements. In this case, the one which gets control of the facility next is chosen by consideration of the following three quantities in order:

(1) Highest control strength

(2) Highest PRIORITY

(3) First to request the facility.

EXAMPLES:

```
RELEASE TERMINAL;
RELEASE LINE;
RELEASE PUMP;
```

d. Testing the Status of a Facility

<facility status> ::= BUSY | NOT BUSY

The status of a facility can be tested for the condition BUSY or NOT BUSY. The facility status can be used in any compound statement as a relation primary.

EXAMPLES:

```
IF TERMINAL BUSY THEN CANCEL;
IF LINE NOT BUSY THEN GO TO LOAD;
WAIT UNTIL TERMINAL NOT BUSY;
```

6. STORES AND ASSOCIATED COMMANDS

STORES are space-shared rather than time-shared global elements and they are assigned a specific storage capacity. As long as there is sufficient storage to accommodate the requesting transaction the request for space is satisfied; otherwise, the transaction waits for the space. A facility may be regarded as a store which has a capacity of one unit only, except for the fact that no interrupt capability is provided for stores.

The following declarations and control statements are associated with manipulating stores:

a. STORE Declaration

```
<store declaration> ::=STORE  <store list>;
<store list> ::= <capacity> <store name>[,<store list>]
<store name>         ::= <name>[(<constant>)]
<capacity>           ::= <constant>
```

STORES are declared at the beginning of the program ahead of the processes. STORES may be declared as one-dimensional arrays.

```
EXAMPLES:     STORE 10 STACK;
              STORE 512 CORE, 10 BUFFER(5);
```

b. ENTER Statement

```
<enter statement>::= ENTER <store name>;|
                     ENTER <store name>, <expression>;
```

The first form is an abbreviation for "ENTER <store name>, 1." The value of the expression is truncated and represents the number of units requested of the store. The transaction will remain at this statement until that number of units becomes available and until all other transactions of greater or equal priority which have been waiting for storage space have been serviced.

EXAMPLES:

```
ENTER STACK;
ENTER CORE, 256;
ENTER CORE, BYTE * LENGTH;
```

c. LEAVE Statement

<leave statement> ::= LEAVE <store name> ; |
LEAVE <store name>,<expression>;

The first form is an abbreviation for "LEAVE <store name>, 1." This statement returns the number of units equivalent to the value of the (truncated) expression.

EXAMPLES:

```
LEAVE STACK;
LEAVE CORE, 128;
LEAVE BUFFER (NODE), LENGTH;
```

d. Testing the Status of a Store

<store status> = FULL | NOT FULL | EMPTY | NOT EMPTY;

The status of a store can be tested for the following conditions:

FULL, NOT FULL, EMPTY, NOT EMPTY.

In combination with other SOL or PL/I statements a variety of compound statements may result.

EXAMPLES:

```
IF SWITCH FULL THEN WAIT PAUSE;
IF SWITCH NOT FULL THEN ENTER SWITCH;
```

7. TRUNKS AND ASSOCIATED COMMANDS

TRUNKS are space-shared global elements similar to STORES. However, in contrast to stores, trunks allow for preemption. As long as there is sufficient storage to accommodate the requesting transaction, the request for space is satisfied without further action. Each transaction holding space in a trunk is assigned a specific holding strength, which may be different from the preemption strength. Thus, a transaction with a low preemption strength once assigned space in a trunk can have a very high holding strength; therefore, preemption of it becomes unlikely.

a. TRUNK Declaration

```
<trunk declaration>::=TRUNK <trunk list>;
<trunk list>::= <capacity> <trunk name>[,<trunk list>]
<trunk name>::= <name>[(<constant>)]
```

TRUNKS are declared at the beginning of the program ahead of the processes. TRUNKS may be declared as one-dimensional arrays. All elements of a TRUNK array assume the same capacity value.

EXAMPLES:

```
TRUNK 96 SWITCH(16);
TRUNK 1000 CORE;
```

b. DEMAND Statement

```
<demand statement>::= DEMAND <trunk name>, <capacity>,
                      <demand strength>, <hold strength>;
<trunk name>::= <name>[(<constant>)] |<name>(<variable>)
```

The demand statement is a request for a number of units of a trunk. If the units requested are available in the trunk, they are assigned to the transaction. Associated with the resource allocation is the hold strength specified in the demand statement.

If the required number of units is not available, then the following takes place:

(1) The number of units needed through preemption is calculated.

(2) The sum of the space held by other transactions at a hold strength less than demand strength of the demanding transaction is determined.

(3) If the total available and preemptable space is sufficient to satisfy the demand, transactions are preempted as required to free enough space. The demanding transaction is then allocated the space with the associated hold strength and continues in sequence. Note that the interrupted transactions are handled according to the setting of their interrupt action indicator.

(4) If there is not enough preemptable space in the trunk, the transaction is queued up on demand strength.

EXAMPLES:

```
DEMAND LINK(15), TRANSM_RATE, 4, 10;
DEMAND CORE(NODE), 512, A, (A + C)/B;
```

c. YIELD Statement

<yield statement> ::= YIELD <trunk name> , <capacity>,
<hold strength>;
<trunk name> ::= <name>[(<constant>)] |<name>(<variable>)

The yield statement releases the specified number of units in the trunk at the specified hold strength. If the number to be released is greater than the number currently held by the transaction at that hold strength, the simulation terminates with an error..

EXAMPLES: YIELD LINES(5), 400, 10;

d. CAPACITY Function

<primary>::=CAPACITY (<trunk name>, <demand strength>)

The capacity function is provided to test the status of a trunk. The capacity function returns the number of units available for a demand of the capacity of the specified demand strength. No resources are allocated and the current state of the trunk is not touched. This function allows the simulation to interrogate the state of the trunk prior to attempting a demand statement upon it.

8. TRANSACTIONS AND ASSOCIATED STATEMENTS

Transactions represent discrete elements "flowing" through the model. They are local to a particular process and may have a number of descriptors (local variables). For example, in a road network simulation a transaction may represent individual vehicles. The properties of these vehicles, such as speed, number of passengers, fuel consumption, etc., are described by the local variables. Each transaction has its own set of local variables. The following statements directly control the creation, disappearance, queuing, or transfer of transactions.

a. Creation of Transactions. At the beginning of simulation there is one transaction present for each process described. Each of these initial transactions starts at time zero and is positioned at the beginning of the process. More transactions may be created by using "start statements."

<start statement>::= NEW TRANSACTION TO <label>;

This statement, when executed, creates a new transaction (whose local variables are the same in number and value as those of the transaction which created it). The new transaction begins executing the program at label while the original transaction continues in sequence.

b. Disappearance of Transactions. Transactions "die" when they execute a cancel statement.

<cancel statement>::=CANCEL;

An implied cancel statement is at the end of every process, so cancel statements need not always be explicitly written. Transactions are also cancelled when they are preempted and the global variable INTERRUPT has been set to CANCEL (see discussion of 'Interrupt').

c. Queuing Of Transactions. Whenever a transaction encounters a blocked resource such as a full store, a busy facility or a full trunk, it automatically enters the queue associated with this resource. Besides these situations the following wait conditions may be programmed for:

(1) WAIT Statements

<wait statement>::=WAIT <expression>;

The expression is truncated, and then this statement advances "TIME" by MAX(0,<expression>), as far as this transaction is concerned. All time delays in a simulated process are, in the last analysis, specified by using wait statements.

EXAMPLES:

```
WAIT 400;
WAIT SIMULATION_TIME;
WAIT (A + B)/C;
```

(2) WAIT UNTIL Statements

<wait-until statement>::=WAIT UNTIL <relation>;

This causes the transaction to freeze at this point until the <relation> becomes true (because of action by other transactions). The relation must not involve expressions which have a random value; e.g., it is not legal to write "WAIT UNTIL PROBABILITY 10" or "WAIT UNTIL A = 1:4 ," etc.

EXAMPLES:

```
WAIT UNTIL SWITCH EMPTY;
WAIT UNTIL TIME > SIMULATION_TIME;
```

d. Transfer Of Transactions

(1) GO TO Statements

<go to statement> ::= GO TO <label>;

This statement is used to transfer to another point in the program; statements are usually executed sequentially.

(2) INTERRUPT Statement

INTERRUPT = WAIT;|CANCEL;|<label>;

INTERRUPT is a global variable which specifies the action to be taken for a preempted transaction. Whenever the interrupt variable has been set, the action for all subsequent preemptions any place within the program is specified unless the interrupt variable is reset.

INTERRUPT = WAIT;

If the interrupted transaction is executing a WAIT statement when interrupted, the wait time is increased by the time which passed during the interrupt. If the interrupted transaction was executing anything other than a WAIT, the transaction is cancelled.

INTERRUPT = CANCEL;

The interrupted transaction is unconditionally cancelled. (Refer to cancel statement).

INTERRUPT = <label>;

The interrupted transaction is started immediately at the statement specified by label and the transaction no longer controls the preempted facility.

9. SPECIAL SOL STATEMENTS

a. PRIORITY Statement. If by coincidence two transactions attempt to do something at precisely the same time, they may be in conflict; that is, they may both want to seize a facility, to change the value of the same global variable, or one may want to change it while the other is using its value. Actually, in such cases of conflict, the simulator does choose a specific order for execution; no two things actually happen at the same instant, as we deal more properly with infinitesimal differences of time between the discrete units. The choice of order is fairly arbitrary except when a difference of priority is specified; in that case, the *transaction with higher priority* (lower value) will be acted on first. Each transaction has a priority, which is initially zero; priority is changed by the statement

PRIORITY = <expression>;

The declaration "integer PRIORITY" is implied at the beginning of each process; i.e., PRIORITY is treated as a local variable. In the present implementation of SOL, the priority must be between 0 and 15.

b. STOP Statement

<stop statement> ::= STOP;

A stop statement causes simulation to terminate immediately, and all transactions cease.

c. Checkpoint - Restart

(1) The BREAKOUT Statement. For long simulation runs it becomes necessary to program checkpoints to save intermediate results for possible later restarts. Checkpoint data are saved whenever a BREAKOUT statement is executed.

<breakout statement> ::= BREAKOUT TO <label>;

If a label is specified, the transaction executing the breakout statement will be restarted at this label at restart time, but will continue its execution after a checkpoint in regular fashion. Checkpoints are numbered sequentially and the simulation may be restarted at any of them. The PARM parameter in the execution card is used to specify the restart point.

(2) The RESTART Statement. The restart statement serves to save values of variables not declared within the SOL syntax. In this way, values of regular PL/1 variables declared with a PL/1 declare statement may also be saved at a checkpoint.

<restart statement> ::= RESTART <variable list>;

The variable list contains only the names of PL/1 variables and not the dimensions of variable arrays.

EXAMPLE:

```
DCL (A,B,C,D) FIXED BIN(15),
    E(5) BIT(3),
    F(0:100,0:2) FLOAT DEC;

    RESTART  A,B,C,D,E,F;
```

d. Collection of Histogram Data. The tabulate statement in conjunction with the table declaration is the vehicle for collecting data to be displayed in histogram format.

(1) TABLE Declaration

```
<min> ::= <constant>
<inc> ::= <constant>
<max> ::= <constant>
<table declaration> ::=
        TABLE(<min> BY <inc> TO <max>) <table name>(<constant>);
```

The table is used in conjunction with the TABULATE statement to collect data for histographical representation. The histogram can be specified by its dimensions, min, inc, max.

```
EXAMPLE:  TABLE (0 BY 100 TO 1000) TIMETABLE(5) ;
```

(2) TABULATE Statements

```
<tabulate statement>::=TABULATE <expression> IN <table name>;
```

The value of the expression is recorded as a statistical observation in the table specified.

```
EXAMPLES:    TABULATE (STARTIME-TIME) IN DIFFTABLE;
             TABULATE TIME IN TIMETABLE;
```

10. COMPOUND AND CONDITIONAL STATEMENTS

Both of those statements are legal in SOL as well as in PL/I. Because of their relative importance and frequent use, they are listed separately.

a. Compound Statements Several statements may be combined into one, as follows:

```
<statement list>::=<statement>; [<statement list>];
<compound statement>::=BEGIN <statement list> END;|
    (<statement list>)
```

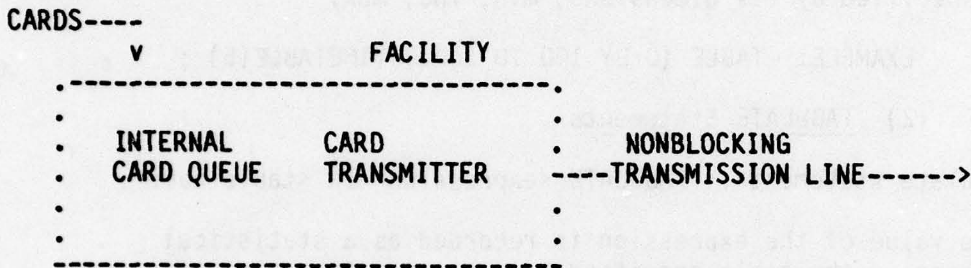
b. Conditional Statement

```
<condition>::= IF <relation> THEN <statement>;|
IF <relation> THEN <unconditional statement> ELSE <statement>;
```

III. SAMPLE MODELS

This section contains a description of nine sample models, including listings of the source language code. Most listings are self-descriptive. However, a more detailed explanation has been provided for MODEL 1E.

1. MODEL 1A: Single Server Queuing Model With Constant Arrival Rate.



PROBLEM: Punched cards are arriving at a card transmitter station with a constant interarrival interval of 36 seconds. The transmitter can handle only one card at a time and needs 40 seconds to process one card. The simulation is to stop after 5 minutes.

2. MODEL 1B: Single Server Queuing Model with Poisson Distributed Arrivals.

PROBLEM: As in Model 1A, except punched cards are arriving poisson distributed with an average arrival rate of 100 cards/minute.

3. MODEL 1C: Single Server Model with Parameterized Input.

PROBLEM: Same as in 1B, except time constants are to be replaced by variables which are to be assigned values from a data set.

4. MODEL 1D: Single Server Queuing Model with Priority

PROBLEM: Same as in 1C, except cards are assigned priorities between 1 and 8 on a random basis. The transmitter is to select cards from the input queue according to its priority strength. A message is to be printed, when card is received.

5. MODEL 1E: Single Server Queuing Model with Preemption.

PROBLEM: Same as in ID, except preempt levels between 1 to 4 are to be assigned randomly. The preempted messages are to be cancelled after a message has been printed.

6. MODEL 1F: Single Server Queuing Model with External Queue

PROBLEM: Same as in 1E, except the STORE resource 'QUEUE' is used to model a physical queue ahead of the transmitter. This QUEUE is used to monitor the queue buildup during the simulation, since the internal queue of the facility is not accessible to the user. Furthermore, a separate process 'CONTROL' is used to read in variable values and control the length of the simulation.

7. MODEL 1G: Network Model - 5 Nodes Fully Connected, but Nonblocking Links.

PROBLEM: Each node is modeled as a combination of a card transmitter as described in 1G and a card receiver of a similar type. The network is fully connected and nonblocking. The originating nodes and the terminating nodes are picked at random. Each message is to be assigned an identification number.

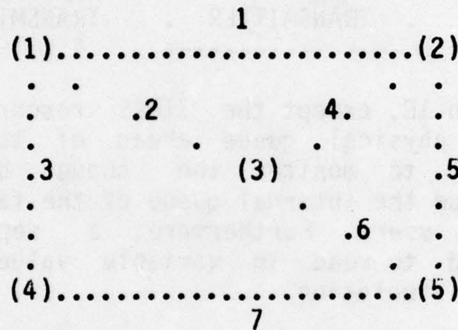
8. MODEL 1H: Network Model with Blocking Links.

PROBLEM: Same as in 1G except that blocking on links is considered. All links are to have eight channels. No alternate routing will be considered. The connecting matrix 'CONN' provides for cross-reference between nodes and links.

	1	2	3	4	5
1	0	1	2	3	0
2	1	0	4	0	5
3	2	4	0	0	6
4	3	0	0	0	7
5	0	5	6	7	0

9. MODEL II: Network Model with Alternate Routing.

PROBLEM: Same as in 1H. Network has the following connectivity (Links are numbered from 1 to 7):



The alternate routing will be of the following type:

Each node has a primary plus two alternate next nodes to choose from for routing a message. The selection will simply be based on the blocking of the links. The modeler may assume that the routing algorithm has been precoded as a function named 'ROUTE' with two arguments representing the current node and the destination node.

<variable>= ROUTE (<orig. node>,<term. node>);

The function will return the next tandem node number of the value '0' if blocking occurs.

```
/* MODEL1A  -  SINGLE SERVER QUEUING MODEL */
/*          WITH UNIFORMLY DISTRIBUTED ARRIVALS */
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
START:  IF TIME > 3000 THEN STOP;
        NEW TRANSACTION TO SEND;
        WAIT 360;
        GO TO START;
SEND:   SEIZE TRANSMITTER;
        WAIT 400;
        CANCEL;
END;

/* MODEL1B  -  SINGLE SERVER QUEUING MODEL */
/*          WITH POISSON DISTRIBUTED ARRIVALS */
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
START:  IF TIME > 3000 THEN STOP;
        WAIT EXPONENTIAL(360);
        NEW TRANSACTION TO START;
        SEIZE TRANSMITTER;
        WAIT 400;
        CANCEL;
END;

/* MODEL1C - SINGLE SERVER/PARAMETERIZED INPUT */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
START:  IF TIME > SIMTIME THEN STOP;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        SEIZE TRANSMITTER;
        WAIT SERTIME;
        CANCEL;
END;

/* MODEL1D  -  SINGLE SERVER WITH PRIORITY HANDLING */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
PROCESS TRANSMIT, T=10, R=1;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
START:  IF TIME > SIMTIME THEN STOP;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        PRIORITY = 1:8;
        PLIBEGIN;
        PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
        PLIEND;
        SEIZE TRANSMITTER;
        WAIT SERTIME;
        CANCEL;
END;
```

```
10 /* MODEL1E  -  SINGLE SERVER QUEUING MODEL */
20 /*                  WITH PREEMPTION        */
30 INTEGER SIMTIME,INTTIME,SERTIME;
40 FACILITY TRANSMITTER;
50 PROCESS TRANSMIT, T=10, R=1;
60 INTEGER STRENGTH;
70 GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
80 INTERRUPT = FINISH;
90 START:  IF TIME > SIMTIME THEN STOP;
100          WAIT EXPONENTIAL(INTTIME);
110         NEW TRANSACTION TO START;
120          PRIORITY = 1:8;
130          STRENGTH = 1:4;
140 PLIBEGIN;
150          PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
160 PLIEND;
170          SEIZE TRANSMITTER,STRENGTH;
180          WAIT SERTIME;
190          CANCEL;
200 FINISH:
210 PLIBEGIN;
220 PUT EDIT('PREEMPTION OCCURRED AT ',TIME) (A(23),F(6)) SKIP;
230 PLIEND;
240 CANCEL;
250 END;
```

EXPLANATION TO MODEL 1E. CODE

STATEMENTS 10, 20.   /* MODEL 1E - SINGLE SERVER QUEUING MODEL*/
/*   WITH PREEMPTION   */

The first two statements contain explanatory text. Since it is bracketted with /* . . . . */ it will be ignored by the translator.

STATEMENT 30.   INTEGER SIMTIME, INTTIME, SERTIME;

This statement declares the global variables SIMTIME, INTTIME, and SERTIME. Within the model these variables will assume the following meaning.

SIMTIME = Simulation time, the time after which the simulation is to be terminated.

INTTIME = Interarrival time for transactions.

SERTIME = Server time, the time a transaction will seize a facility until it is served.

STATEMENT 40. FACILITY TRANSMITTER;

This statement declares one nonsubscribed facility with the name TRANSMITTER.

STATEMENT 50. PROCESS TRANSMIT, T=10, R=1;

This statement declares the process with the name TRANSMIT. T=10 specifies that not more than 10 transactions will be active at any one time during the simulation. An active transaction is a transaction which has been created and not yet cancelled.

R=1 specifies that no transaction will use more than one resource. During model checkout, these parameters should be kept to a minimum to optimize core utilization.

STATEMENT 60. INTEGER STRENGTH;

This statement declares STRENGTH as a local variable within the process.

STATEMENT 70. GET FILE(CARD) LIST(SIMTIME, INTTIME, SERTIME);

This statement is a PL/1 statement which has been inserted into the SOL route to read the file named 'CARD' and assign the first three numerical values to the global variables SIMTIME, INTTIME, SERTIME.

STATEMENT 80. INTERRUPT = FINISH;

This statement specifies that any preempted transaction is to be sent to Label 'FINISH'.

STATEMENT 90. START: IF TIME > SIMTIME THEN STOP;

This compound statement, identified by the label 'START', tests the global variable SIMTIME against the built-in global variable TIME. TIME is a reserved word within SOL and represents the current time of the simulation. If TIME exceeds the value for SIMTIME, the simulation will be terminated, as specified by the STOP statement.

STATEMENT 100. WAIT EXPONENTIAL(INTTIME);

This statement specifies that the transaction is to be placed into the wait queue for the time specified by the built-in function EXP. The EXP function will sample a value from an exponential distribution with the average value INTTIME.

STATEMENT 110. NEW TRANSACTION TO START;

This statement specifies that a new transaction with the same local variables is to be created and to be sent to the label 'START'. The original transaction will continue to run until it encounters a wait status. Then the new transaction will start executing.

STATEMENT 120. PRIORITY = 1:8;

This statement assigns a random integer between 1 and 8 to the built-in local variable 'PRIORITY'. The local variable PRIORITY is used by the system to resolve any conflicts between transactions requiring the same action at the same time. In this case, it will control the seizing of the facility TRANSMITTER by transactions which have entered a queue because the facility was busy.

STATEMENT 130. STRENGTH = 1:4;

This statement assigns an integer value between 1 and 4 to the local variable STRENGTH.

STATEMENTS 140 to 160. PLIBEGIN; PUT EDIT ('CARD RECEIVED AT', TIME) (A(17), F(6)) SKIP; PLIEND;

These three statements represent a PL/1 block which was inserted to send a message to the SYSPRINT file. The PL/1 PUT statement has been bracketed by PLIBEGIN and PLIEND. In this way the entire PL/I block is bypassed by the translator and the associated text inserted unaltered.

STATEMENT 170. SEIZE TRANSMITTER, STRENGTH;

One of the following actions takes place:

a. If the facility TRANSMITTER is not busy, the transaction simply seizes the facility and marks it busy. An entry is placed into the log file.

b. If the facility is busy with a transaction of holding strength equal to or higher than the local variable STRENGTH of the calling transaction, the calling transaction enters the wait queue for this facility.

c. If the facility is busy with a transaction of lower holding strength, this transaction is preempted and sent to the action label FINISH as specified in the INTERRUPT statement.

STATEMENT 180. WAIT SERTIME;

The transaction encountering this statement will enter the wait queue for the period specified by the value of SERTIME. The next transaction in the time queue will then start executing.

STATEMENT 190. CANCEL;

This statement will cause all resources the transaction is using to be freed and the transaction to be deactivated. Corresponding entries are made in the log file.

STATEMENT 200. FINISH;

This is a simple label statement that has been specified as the action label for an interrupt.

STATEMENTS 210 to 230. PLIBEGIN; PUT EDIT ('PREEMPTION OCCURRED AT',TIME) (A(23), F(6)) SKIP; PLIEND;

These three statements represent a PL/I block, similar to statements 140 to 160, which will cause a message to be sent to the SYSPRINT file whenever a transaction is preempted.

STATEMENT 240. CANCEL;

This statement will deactivate the transaction.

STATEMENT 250. END;

The END statement identifies the end of the process and is ignored by the transactions.

```
/* MODEL1F  -  SINGLE SERVER QUEUING MODEL */
/*              WITH EXTERNAL QUEUE        */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER;
STORE 1000 QUEUE;
PROCESS CONTROL,T=1,R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=10, R=2;
INTEGER STRENGTH;
INTERRUPT = FINISH;
START;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        PRIORITY = 1:8;
        STRENGTH = 1:4;
        PUT EDIT ('CARD RECEIVED AT ',TIME)(A(17),F(6)) SKIP;
        ENTER QUEUE;
        SEIZE TRANSMITTER, STRENGTH;
        WAIT SERTIME;
        CANCEL;
FINISH:  PUT EDIT('PREEMPTION OCCURRED AT ',TIME) (A(23),F(6)) SKIP;
CANCEL;
END;

/* MODEL1G  -  SINGLE SERVER QUEUING MODEL WITH EXTERNAL QUEUE */
INTEGER SIMTIME,INTTIME,SERTIME;
FACILITY TRANSMITTER(5),RECEIVER(5);
STORE 1000 SENDQUEUE(5), 1000 RECEIVEQUEUE(5);
PROCESS CONTROL, T=1, R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=10, R=4;
INTEGER STRENGTH,ORIG,DEST,NUMBER;
INTERRUPT = FINISH;
NUMBER=0;
START;
        WAIT EXPONENTIAL(INTTIME);
        NEW TRANSACTION TO START;
        NUMBER = NUMBER+1;
        PRIORITY = 1:8;
        STRENGTH = 1:4;
        ORIG = 1 : 5;
        DEST = 1 : 5;
PLIBEGIN;
        PUT EDIT ('CARD ',NUMBER,' RECEIVED AT NODE ',ORIG,
        ' AT TIME ',TTIME) (A(5),F(4),A(18),F(2),A(9),F(6)) SKIP;
PLIEND;
            ENTER SENDQUEUE(ORIG);
```

```
            SEIZE TRANSMITTER(ORIG),STRENGTH;
            ENTER RECEIVEQUEUE(DEST);
            SEIZE RECEIVER(DEST),STRENGTH;
            WAIT SERTIME;
            CANCEL;
FINISH;
PLIBEGIN;
   PUT EDIT('CARD ',NUMBER,' PREEMPTED AT TIME ',TIME)
         (A(5),F(4),A(19),F(6)) SKIP;
PLIEND;
CANCEL;
END;
```

```
INPUT PARAMETER LIST FOR STATISTICS.    SOL.DATA(STATIN)

                  NO
                   0,
                  NO
                  NO
```

CLASS(Z) OUTPUT OF STATISTICS STEP 'SOL(S)' FOR MODEL1G

NAME OF FACILITY	TIME	FRACTION OF TIME IN USE
TRANSMITTER (1)	20866	0.0958
TRANSMITTER (2)	20866	0.1114
TRANSMITTER (3)	20866	0.1534
TRANSMITTER (4)	20866	0.1342
TRANSMITTER (5)	20866	0.2210
RECEIVER (1)	20866	0.1725
RECEIVER (2)	20866	0.1279
RECEIVER (3)	20866	0.0958
RECEIVER (4)	20866	0.1150
RECEIVER (5)	20866	0.1725

NAME OF STORE	TIME	CAPCTY	MAX USD	TOTAL OCCP	AVG UTL
SENDQUEUE (1)	20866	1000	1	2000	0.0001
SENDQUEUE (2)	20866	1000	2	2324	0.0001
SENDQUEUE (3)	20866	1000	1	3200	0.0002
SENDQUEUE (4)	20866	1000	1	2800	0.0001
SENDQUEUE (5)	20866	1000	2	4933	0.0002
RECEIVEQUEUE(1)	20866	1000	2	3655	0.0002
RECEIVEQUEUE(2)	20866	1000	1	2669	0.0001
RECEIVEQUEUE(3)	20866	1000	1	2000	0.0001
RECEIVEQUEUE(4)	20866	1000	1	2400	0.0001
RECEIVEQUFUE(5)	20866	1000	2	4222	0.0002

CLASS(Y) OUTPUT OF MODEL1G

```
CARD       1 RECEIVED AT NODE  1 AT TIME     470
CARD       1 RECEIVED AT NODE  3 AT TIME     716
CARD       1 RECEIVED AT NODE  5 AT TIME     962
CARD       1 RECEIVED AT NODE  5 AT TIME    1051
CARD       1 RECEIVED AT NODE  2 AT TIME    1185
CARD       1 RECEIVED AT NODE  4 AT TIME    1525
CARD       1 RECEIVED AT NODE  4 AT TIME    2261
CARD       1 RECEIVED AT NODE  5 AT TIME    3501
CARD       1 RECEIVED AT NODE  4 AT TIME    3855
CARD       1 RECEIVED AT NODE  3 AT TIME    4066
CARD       1 RECEIVED AT NODE  1 AT TIME    4601
CARD       1 RECEIVED AT NODE  3 AT TIME    5137
CARD       1 RECEIVED AT NODE  3 AT TIME    5987
CARD       1 RECEIVED AT NODE  5 AT TIME    6319
CARD       1 RECEIVED AT NODE  3 AT TIME    6869
CARD       1 RECEIVED AT NODE  5 AT TIME    7065
CARD       1 RECEIVED AT NODE  5 AT TIME    9217
CARD       1 RECEIVED AT NODE  2 AT TIME    9835
CARD       1 RECEIVED AT NODE  5 AT TIME    9957
CARD       1 RECEIVED AT NODE  2 AT TIME   10104
CARD       1 RECEIVED AT NODE  3 AT TIME   10282
CARD       1 RECEIVED AT NODE  3 AT TIME   11004
CARD       1 RECEIVED AT NODE  2 AT TIME   11465
CARD       1 RECEIVED AT NODE  4 AT TIME   12102
CARD       1 RECEIVED AT NODE  2 AT TIME   12447
CARD       1 RECEIVED AT NODE  1 AT TIME   13239
CARD       1 RECEIVED AT NODE  1 AT TIME   14493
CARD       1 RECEIVED AT NODE  4 AT TIME   14932
CARD       1 RECEIVED AT NODE  5 AT TIME   14933
CARD       1 RECEIVED AT NODE  4 AT TIME   15340
CARD       1 RECEIVED AT NODE  3 AT TIME   15512
CARD       1 RECEIVED AT NODE  4 AT TIME   16097
CARD       1 RECEIVED AT NODE  5 AT TIME   16307
SIMULATION TERMINATED - I/O ERROR
```

CONTENTS OF FILE SOL.DATA(GOIN): 2000, 500, 400

```
/* MODEL1H - 5-NODE FULLY CONNECTED NETWORK WITH BLOCKED LINKS */
INTEGER SIMTIME,INTTIME,SERTIME,NUM;
PLIBEGIN;
DCL CONN(5,5) FIXED BIN(31);
PLIEND;
FACILITY TRANSMITTER(5),RECEIVER(5);
STORE 1000 SENDQUEUE(5), 1000 RECEIVEQUEUE(5), 8 LINK(10);
PROCESS CONTROL, T=1, R=0;
GET FILE(CARD) LIST(SIMTIME,INTTIME,SERTIME,CONN);
WAIT SIMTIME;
STOP;
END;
PROCESS TRANSMIT, T=50, R=5;
INTEGER STRENGTH,ORIG,DEST,NUMBER;
INTERRUPT = FINISH;
NUM=0;

START:
       WAIT EXPONENTIAL(INTTIME);
       NEW TRANSACTION TO START;
       NUMBER,NUM = NUM+1;
       PRIORITY = 1:8;
       STRENGTH = 1:4;
       ORIG = 1 : 5;
RET:   DEST = 1 : 5;
       IF DEST = ORIG THEN GO TO RET;
     PUT EDIT ('CARD ',NUMBER,' RECEIVED AT NODE ',ORIG,' AT TIME ',
TIME ', TO =',DEST) (A(5),F(4),A(18),F(2),A(9),F(6),A(5),F(2)) SKIP;
       ENTER SENDQUEUE(ORIG);
       SEIZE TRANSMITTER(ORIG),STRENGTH;
       ENTER LINK(CONN(ORIG,DEST));
       ENTER RECEIVEQUEUE(DEST);
       SEIZE RECEIVER(DEST),STRENGTH;
       WAIT SERTIME;
        CANCEL;
FINISH:   PUT EDIT('CARD ',NUMBER,' PREEMPTED AT TIME ',TIME)
          (A(5),F(4),A(19),F(6))  SKIP;
CANCEL;
END;
```

Input Dataset SOL.DATA(GOIN):

```
20000,100,500,
0,1,2,3,4,
1,0,5,6,7,
2,5,0,8,9,
3,6,8,0,10,
4,7,9,10,0,
```

BIBLIOGRAPHY

1. D. E. Knuth and J. L. McNeley, "SOL - A Symbolic Language for General Purpose Systems Simulation ," IEEE Transactions on Electronic Computers, IC-13, No. 5 (Aug 1964) pp 401-408.

2. R&D Technical Report ECOM-3085 (AD-850159L), "MALLARD Traffic Simulation, Results and Analysis, Final Report," James A. Armstrong and Horst E. Ulfers, Feb 1969.

3. J. Armstrong, H. Ulfers, D. Miller, H. Page, "SOLPASS - A Simulation Oriented Language Programming and Simulation System," Proceedings of the Third Conference on Applications of Simulation, Dec 1969.

4. R&D Technical Report ECOM-0043-F, "SOL Compiler Design," H.C. Page, D.J. Miller (Patterson & Smith Inc), Feb 1968.

5. Horst E. Ulfers, "PACKNET - A Packet Switch Network Simulator," Proceedings of the 1975 ICC, June 1975.

6. C. G. Guffee and H. E. Ulfers , "SOL-370," Proceedings of the 1975 Summer Computer Simulation Conference, July 1975, pp 1-11.

7. DCEC TN 10-78, "SOL-370 User's Guide," Horst E. Ulfers, July 1978.

DISTRIBUTION LIST

STANDARD:

R100 - 2
R102/R103/R103R - 1
R102M - 1
R102T - 9
R104 - 1
R110 - 1
R123 - 1
R124A - 1

R200 - 1
R300 - 1
R400 - 1
R500 - 1
R700 - 1
R800 - 1
NCS-TS - 1

205 - 20

DCA-EUR - 1 (Defense Communications Agency European Area
ATTN: Technical Director
APO New York 09131)

DCA-PAC - 1 (Defense Communications Agency Pacific Area
ATTN: Technical Director
Wheeler AFB, HI 96854)

USDCFO - 1 (Chief, USDCFO/US NATO
APO New York 09667)

SPECIAL:

R830 - 100