MEMORANDUM
RM-3310-PR
NOVEMBER 1962

SIMSCRIPT: A SIMULATION PROGRAMMING LANGUAGE

H. M. Markowitz, B. Hausner and H. W. Karr

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-3310-PR
NOVEMBER 1962

SIMSCRIPT: A SIMULATION PROGRAMMING LANGUAGE

H. M. Markowitz, B. Hausner and H. W. Karr

This research is sponsored by the United States Air Force under Project RAND — Contract No. AF 49(638)-700 — monitored by the Directorate of Development Planning, Deputy Chief of Staff, Research and Technology, Hq USAF. Views or conclusions contained in this Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force. Permission to quote from or reproduce portions of this Memorandum must be obtained from The RAND Corporation.

The RAND Corporation
1700 MAIN ST • SANTA MONICA • CALIFORNIA

PREFACE

The SIMSCRIPT Simulation Programming Language described in this manual is a direct descendant of two earlier systems. Its immediate predecessor, SPS–1 (Simulation Programming System–1), was developed at The RAND Corporation. SPS–1, in turn, was an outgrowth of GEMS (General Electric Manufacturing Simulator), which was previously developed by H. Markowitz at General Electric Manufacturing Services. Since neither GEMS nor SPS–1 was distributed publicly, it seems appropriate at this time to acknowledge the help of certain individuals who contributed to these parent systems, as well as to acknowledge those who have contributed directly to SIMSCRIPT.

GEMS benefited greatly from the excellence and diligence of Morton Allen, who did most of the GEMS programming. Regretfully, we must forebear from listing a number of other contributors at General Electric (particularly several each from Manufacturing Services, Ordnance Department, Circuit Protective Devices Department, and Lynn Computations Operation).

Jack Little of RAND contributed to early discussions of SPS–1. Richard Conway of Cornell University, who spent several months at RAND during which time he programmed extensively in SPS–1, contributed to the design of SIMSCRIPT.

The three authors whose names appear on this manual constituted the design team for SIMSCRIPT. H. Markowitz acted as chairman and had final responsibility for the logical design of the system. B. Hausner was solely responsible for the programming of both the SPS–1 and SIMSCRIPT Translators. H. Karr wrote the present SIMSCRIPT manual.

The SIMSCRIPT system was developed under the auspices of Project RAND to provide more efficient ways of preparing simulation programs of direct benefit to RAND projects, and indirectly, to benefit those many other organizations using simulation. A copy of the SIMSCRIPT translator for the 709/7090 computer can be obtained by sending a blank tape to the Computer Sciences Department of RAND.

While SIMSCRIPT was being written, others elsewhere were also experimenting with simulation programming techniques. Among systems worth noting are SIMPAC (Mike Lackner, System Development Corporation, Santa Monica), GPSS (Geoffrey Gordon, IBM, White Plains), and CSL (Laski, Esso Petroleum, U. K.; and Buxton, IBM, U. K.).

SUMMARY

In a variety of fields, simulation with digital computers has proved to be a valuable management–analysis technique. One obstacle, however, has been the substantial time needed to program simulations of even moderate complexity. The SIMSCRIPT system described in this manual was developed to meet the need to reduce programming time. It also provides increased flexibility in modifying such models in accordance with the findings of preliminary analysis and other circumstances.

Although SIMSCRIPT may also be used as a computer language for non–simulation problems, our discussion in this Summary and our main emphasis in the manual which follows, is in terms of its application to simulation. The purpose of this manual is to enable a person already familiar with computer programming to apply SIMSCRIPT to his task, by furnishing him the necessary information including detailed instructions and forms.

Every digital simulation involves a numerical description of the "Status" of the system to be simulated. This "Status" is modified as "Events" occur at various points in simulated time. The SIMSCRIPT view of Status and Events may be schematized as follows:

$$\text{STATUS} \quad \left\{ \begin{array}{l} \text{Entities} \\ \qquad \text{Temporary} \\ \qquad \text{Permanent} \\ \text{Attributes} \\ \text{Sets} \end{array} \right.$$

$$\text{EVENTS} \quad \left\{ \begin{array}{l} \text{Exogenous} \\ \text{Endogenous} \end{array} \right.$$

As of any particular moment in simulated time, the Status of the system being portrayed is described in terms of what "Entities" exist, what the current values of their "Attributes" are, what "Sets" they belong to, and what "Sets" they own. Each kind of Event may occur repeatedly, and at any desired points in simulated time. Some may occur "exogenously," impelled by forces outside the simulated process. Others may occur "endogenously," caused by preceding Events within the simulation.

A standardized definition form is used to tell the SIMSCRIPT translator the names of the different types of Entities, Attributes, and Sets to be distinguished in a particular simulation. Various other kinds of inform–

ation concerning these Entities, Attributes and Sets are also indicated on the definition form (such as the number of words of computer memory required to store the Attributes of "Temporary" Entities; the integer or floating point mode of Attributes; and the organization of Sets, e. g., LIFO or FIFO).

In SIMSCRIPT, a sub—program must be written for each different kind of event, describing how it changes Status or causes future Events. These routines are written in the SIMSCRIPT source language, which provides commands (such as CREATE, DESTROY, CAUSE, CANCEL, REMOVE, FILE, FIND MAX, etc.) particularly suited to the needs of simulation, as well as more conventional arithmetic, control, and input—output commands.

Finally, SIMSCRIPT contains a "Report Generator" which permits the user to specify the form, content, and the repetition of printed output on a layout form. From this an output routine is generated without further programming.

CONTENTS

FIGURES

TABLES

Chapter 1

BASIC CONCEPTS

THE USE OF SIMULATION

In recent years, computer simulation of real systems has become a valuable aid to managers and analysts in such diverse fields as manufacturing, logistics, economics, transportation, and military operations. Computer programs have been developed to simulate the behavior through time of manufacturing systems, logistics systems, transportation systems, and the like. These computer simulations are used to trace out the performance of alternate system configurations guided by alternate sets of decision rules under various conditions.

In the simulation of manufacturing systems, for example, the computer is told the physical structure of the manufacturing facility—the number of machines of each kind, the number of men of each kind on each shift, which men can work on which machines, and so on. It is also told the rules the system operates under, such as the dispatching rules for determining which order to work on first, and overtime rules for determining when men will work extra hours. Finally the computer is told the work load to be processed through the plant, including both the orders waiting at the start of simulation and the orders received during the course of simulated time. With this information the computer runs the orders through the plant, routing them from one machine to the other according to specified routing information, and making them wait their turn for men and machines already occupied. During the course of simulated time the computer keeps track of how well the simulated system performs, according to such measures as machine utilization, labor utilization, and the time required for orders to be completed.

Similarly, the computer can be told (partly through program, partly through data) how to simulate the movement of motor vehicles or aircraft through a transportation system, the processing of supply and maintenance requests through a logistics system, the changing water levels and rates of flow through a hydroelectric system, and the like.

These simulated systems can then be operated any number of times, with variations in the load to be processed, the decision rules to be followed, or the configuration of the physical system itself.

Simulating the dynamics of alternate systems under a variety of conditions often uncovers potential difficulties that traditional static analyses are incapable of finding. If the real system in question is large, complex, and costly, the use of a computer as a "pilot plant" or "wind tunnel" is inexpensive compared to the cost of a trial–and–error approach with the real system.

THE PURPOSE OF SIMSCRIPT

Although simulation has proven to be a valuable tool, it has suffered from the fact that it usually takes several times as long to develop the computer program as it does to formulate the basic simulation model. Many problems to which simulation was applicable in principle could not in fact use it because decisions had to be made before computer programs could be developed.

Fortunately, experience now confirms that much of the time spent in both logical formulation and actual programming is spent on operations that are often similar from one simulation problem to the next. Thus there is a clear opportunity and need for a programming system specially adapted to the problems of writing simulation programs. SIMSCRIPT was designed to answer this need.

Any digital simulation consists of a numerical description of the "Status" of the simulated system. This Status is modified at various points in simulated time which may be called "Events." SIMSCRIPT provides a standardized definition–form for specifying the Status description. It also automatically provides a main timing routine to keep track of simulated time and the occurrence of Events. An "Event Routine" is then written for each kind of Event, describing how the Status is to change. The SIMSCRIPT source language is specifically designed to facilitate the formulation and programming of these Event Routines.

In particular, the following operations may each be accomplished by a single source–language statement: the allocation or return of storage space for temporary variables, the filing of items into or out of "sets," the accumulation of information across simulated time, the summarization of information at a point in time, and the finding of minimums or maximums over collections of items meeting specified conditions. Additional features include subscripted subscripts to any level, and a memory layout philosophy which affords considerable flexibility in making program modifications and which permits both source and object programs to be "dimension–free." A "Report Generator" is also provided which permits the user to specify the form, content, and repetition of printed output on a layout form from which an output routine is generated without further programming.

Although SIMSCRIPT was developed for simulation problems, and the present exposition is presented in terms of simulation problems, SIMSCRIPT is actually a general programming system that is also readily usable for non–simulation problems.

STATUS DESCRIPTION

In SIMSCRIPT, the "Status" of the simulated system is defined in terms of what are called Entities, Attributes of Entities, and Sets of Entities.

Any type of unit to be independently identified in the simulation, such as a truck, a species of animal, a chair, or a bank loan, is called an "Entity."

Each type of Entity is in turn described by enumerating its particular "Attributes." The Attributes of a truck might include its initial cost, its payload, and its operating cost per mile; those of a bank loan might include the amount due, the due date, and the interest rate.

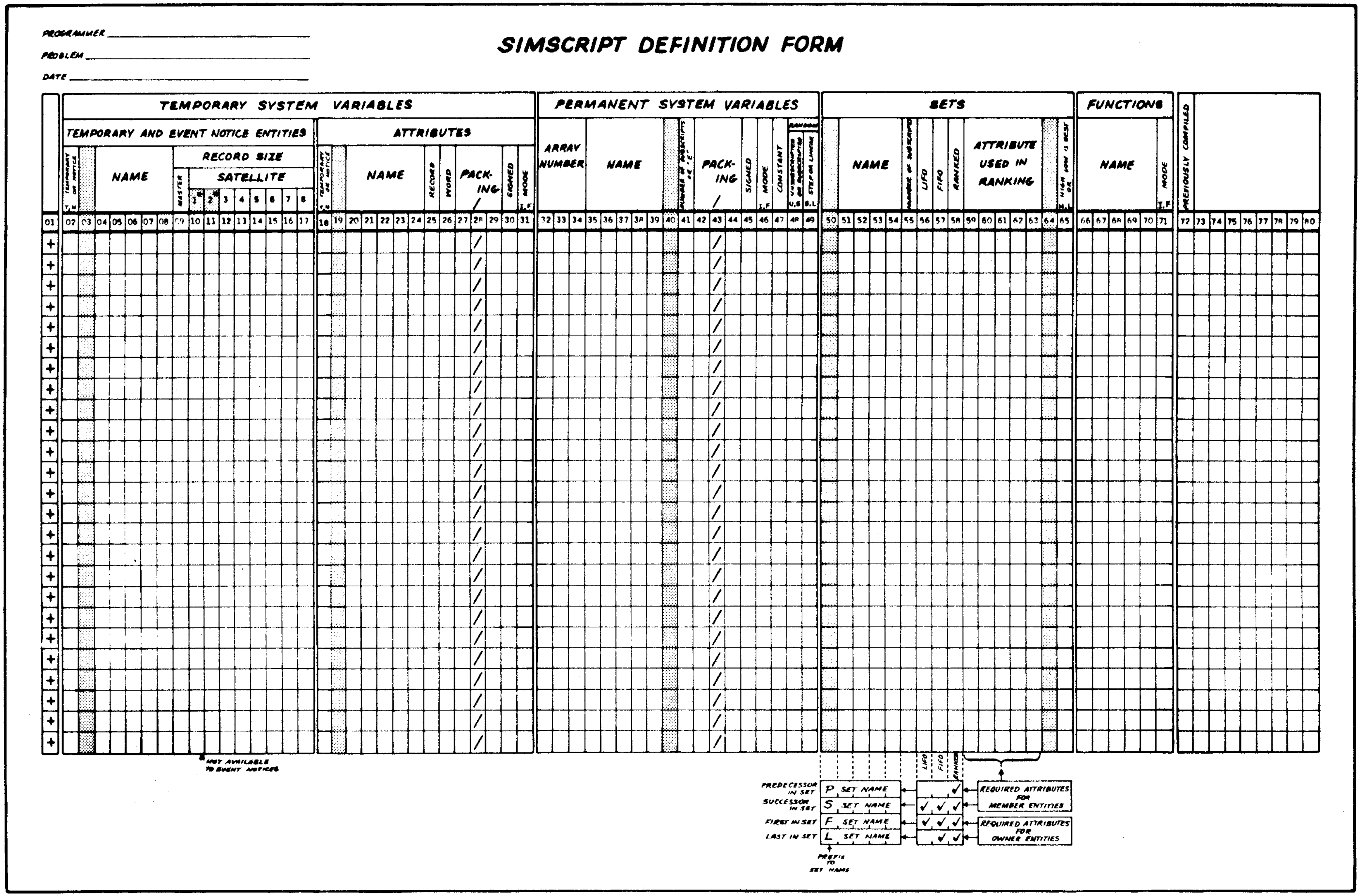
A Status description may comprise any number of different types of Entities; there can also be any number of Entities of a particular type. Entities are considered to be of the same type if their Attribute names are identical; the values of these Attributes may of course be different. Two bank loans, for example, may have different due dates, although both have an Attribute called "due date."

Whether or not something is to be considered an Entity or an Attribute of an Entity depends on whether it is to be independently identified in the simulation. If it is desired to keep track of each unit of a particular spare part, each unit must appear in the Status description as an Entity. If there is no need to distinguish among units, they might appear as "stock on hand," which could be an Attribute of some other type of Entity, such as "stock number."

Interrelationships among individual Entities may be depicted in the Status description by grouping the Entities into "Sets;" for example: "passengers holding reservations for flight 72," or "requisitions on backorder at depot 3." Entities may be members of Sets and also owners of Sets. For example, "passengers" are members of a Set belonging to "flight 72," and "requisitions" are members of a Set belonging to "depot 3." An Entity may belong to any number of Sets and may own any number of Sets. In SIMSCRIPT, Entities may be readily inserted or removed from Sets on a "first-in-first-out" or a "last-in-first-out" basis, or on a "ranked" basis by which the Entity's ranked position is determined by the value of one of its Attributes.

DEFINITION OF ENTITIES, ATTRIBUTES, AND SETS

The procedure for specifying Status consists of defining each different type of Entity, Attribute, and Set on the SIMSCRIPT Definition Form shown in Fig. 1.



PROGRAMMER ______

PROBLEM ______

DATE ______

SIMSCRIPT DEFINITION FORM

TEMPORARY SYSTEM VARIABLES

TEMPORARY AND EVENT NOTICE ENTITIES: TEMPORARY OR NOTICE (T, N) — NAME — RECORD SIZE: MASTER; SATELLITE 1* 2* 3 4 5 6 7 8

ATTRIBUTES: TEMPORARY OR NOTICE (T, N) — NAME — RECORD — WORD — PACKING / — SIGNED — MODE (I, F)

PERMANENT SYSTEM VARIABLES: ARRAY NUMBER — NAME — NUMBER OF SUBSCRIPTS OR "E" — PACKING / — SIGNED — MODE (I, F) — CONSTANT — RANDOM: UNSUBSCRIPTED OR SUBSCRIPTED (U, S); STEP OR LINEAR (S, L)

SETS: NAME — NUMBER OF SUBSCRIPTS — LIFO — FIFO — RANKED — ATTRIBUTE USED IN RANKING — HIGH LOW IS BEST OR (H, L)

FUNCTIONS: NAME — MODE (I, F)

PREVIOUSLY COMPILED

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

* NOT AVAILABLE TO EVENT NOTICES

		LIFO	FIFO	RANKED	
PREDECESSOR IN SET	P SET NAME			✓	REQUIRED ATTRIBUTES FOR MEMBER ENTITIES
SUCCESSOR IN SET	S SET NAME	✓	✓	✓	
FIRST IN SET	F SET NAME	✓	✓	✓	REQUIRED ATTRIBUTES FOR OWNER ENTITIES
LAST IN SET	L SET NAME		✓	✓	

PREFIX TO SET NAME

Fig. 1 — SIMSCRIPT Definition Form (reduced)

Although such differentiation is not essential, certain computing and programming economies can be obtained by distinguishing among different classes of Entities called Temporary Entities, Event Notice Entities, Permanent Entities, and the simulated System, which also may be thought of as an Entity.

Temporary Entities, Event Notices, and their Attributes are defined in the first panel of the Definition Form. Permanent Entities, their Attributes, and Attributes of the System are defined in the second panel. Each kind of Set to be included in the simulation is defined in the third panel. The fourth panel is for defining the names of special Functions. Space is provided for specifying the name by which each type of Entity, Attribute, or Set will be referred to in the source program. Space is also provided for specifying such things as the size of memory records, number of subscripts, sign convention, etc., as may be appropriate for particular definitions.

The meanings of the various Definition Form card fields, and the rules for making the various entries, are illustrated in the example program given in Chapter 3, and are discussed in detail in Chapter 13.

EVENT ROUTINES AND TIMING

A separate subprogram must be written for each different kind of Event, describing how it changes the Status. There may be any number of different kinds of Events, and a particular kind of Event may occur repeatedly and at any desired point in simulated time. For example, one Event routine may describe how Status changes every time a requisition is received at a supply depot. Another may describe how Status changes every time a shipment of material is received at a supply depot.

Each kind of Event is enumerated in what is called an "Events List." Based on this Events List, SIMSCRIPT automatically generates a main Timing routine which keeps track of simulated time and calls the various Event routines in the proper sequence.* In SIMSCRIPT simulation programs, time is advanced by variable increments rather than being broken into a sequence of fixed increments. Therefore, Events may occur at any desired point in simulated time. When the execution of a particular Event routine is finished, simulated time is immediately advanced to the time of the next most imminent Event, whether it be seconds, hours, or days away, and the appropriate Event Routine is automatically called. The intervening time periods when no Status changes occur are skipped. The

*If a non-simulation program is to be written in SIMSCRIPT, the generation of the Timing routine can be suppressed and a control routine called MAIN must be written into the source program to replace it. (See Chapter 9).

SIMSCRIPT Timing routine permits the occurrence of both "Endogenous Events" (those caused by previous Events within the simulation) and "Exogenous Events" (those introduced from outside the simulation by means of an "Exogenous Event Tape").

For purposes of discussion, the various operations performed by Event routines (or by subroutines on which they call) may be grouped under the following headings:

(1) Changing the Status of individual Entities

(2) Causing or cancelling future Events

(3) Executing decision rules

(4) Accumulating and analyzing results

(5) Printing results

CHANGING THE STATUS OF AN ENTITY

Since Status is defined in terms of Entities, Attributes, and Sets, an Entity can change its Status in only three ways:

(1) It can come into or go out of existence (i. e., be created or destroyed);

(2) It can change an Attribute value; or

(3) It can change a Set membership.

SIMSCRIPT provides a number of commands especially adapted to making these changes.

Creating or Destroying an Entity

To create a record in memory for a new Entity that is just coming into existence (for example, a new bank loan) it is necessary merely to write a single statement, such as "CREATE LOAN." This record will be in whatever configuration was specified on the Definition Form. If the bank loan is going out of existence, the statement "DESTROY LOAN" will zero out the record of its Attributes and make the storage space available to subsequent "CREATE" statements. Entities that may be created and destroyed are either "Temporary Entities" or "Event Notices."

Changing An Attribute Value

SIMSCRIPT uses a "LET" statement to change an Attribute value arithmetically. For example, a statement like the following could set the current stock of an item equal to the previous stock plus receipts:

```
        LET STOCK(ITEM) = STOCK(ITEM) + RCPTS(ITEM)
```

If this computation were to be done for each item on a list, the statement could read:

```
        LET STOCK(ITEM) = STOCK(ITEM) + RCPTS(ITEM), FOR EACH
                          ITEM OF LIST
```

In these examples, "STOCK" and "RCPTS" are Attributes of "ITEM," and "LIST" is a Set of ITEMS.

Changing a Set Membership

SIMSCRIPT provides special statements such as "FILE ITEM IN LIST" or "REMOVE FIRST ITEM FROM LIST" for transferring Entities into and out of Sets.

While the preceding examples are not exhaustive, they demonstrate that in changing the Status of an Entity, a single SIMSCRIPT statement is sufficient to create or destroy it, change an Attribute value, or change a Set membership.

CAUSING AND CANCELLING EVENTS

To help keep track of Events, SIMSCRIPT provides a special kind of Temporary Entity called an "Event Notice." Event Notices have the same properties as Temporary Entities: they may be created and destroyed, have Attributes, and be members and owners of Sets. The programming steps for causing a future Event are:

(1) Creating an "Event Notice" for the particular Event;

(2) Posting the values of its Attributes, as required; and

(3) Scheduling its occurrence.

For example, if an order is shipped to a base, its arrival following a transit–time delay might be caused by the following statements:

```
                CREATE ARRVL

                STORE ORDER IN ITEM(ARRVL)

                STORE BASE IN DESTN(ARRVL)

                CAUSE ARRVL AT TIME + TRANT
```

Prior to its occurrence, this Event could be cancelled by the statement "CANCEL ARRVL."

EXECUTING DECISION RULES

In Event routines, it is usually necessary to perform tests or computations in order to decide how Status should change and what future Events should be caused or cancelled. For making simple decisions, an assortment of "IF" and "GO TO" statements are provided. In addition, SIMSCRIPT provides "FIND MAX," "FIND MIN," and "FIND FIRST" statements. For example,

```
        FIND LEVEL = MIN OF STOCK(ITEM, BASE), FOR ITEM =
                     (1)(N), FOR EACH BASE OF ZI, WITH
                     (PRICE(ITEM))GR(CAT(2))
```

In cases where more complex decision rules are desired, the source language provides a full complement of arithmetic and control statements so that any desired algorithm may be programmed.

ACCUMULATING AND ANALYZING RESULTS

Data describing simulation results may be accumulated and analyzed by progressively changing the values of certain Attributes through standard arithmetic and control operations. However, some accumulation and analysis operations are required so often that special commands are provided.

The integral of an Attribute value may be accumulated over time by an ACCUMULATE statement, such as:

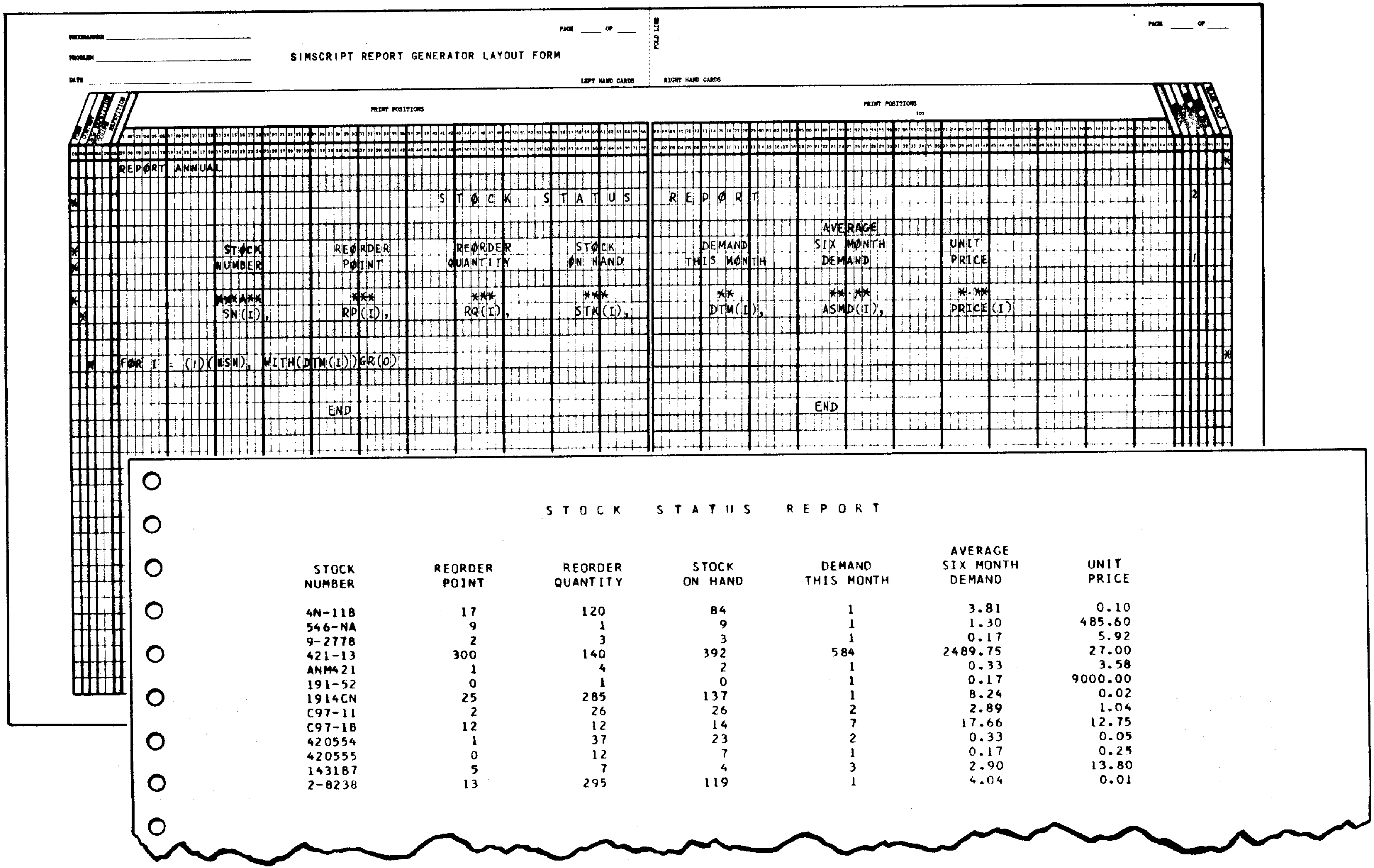
```
ACCUMULATE STOCK(ITEM) INTO CUMSK(ITEM) SINCE LAST(ITEM)
```

Certain frequently desired statistical quantities may be computed by a COMPUTE statement, such as:

```
COMPUTE M, S, V = MEAN, STD-DEV, VARIANCE OF PRICE(ITEM),
                  FOR EVERY ITEM OF JOB
```

PRINTED OUTPUT

Printed output in SIMSCRIPT programs normally comes from the "SIMSCRIPT Report Generator," which generates output routines based on the contents of the "Report Generator Layout Form" in Fig. 2. The form and content of the printed output are specified on the Layout Form by what are called "Form Lines" and "Content Lines." Any desired text may be specified in a Form Line by writing the characters into the print positions where it is desired to have them appear. Numerical fields for printing the values of Attributes are specified by inserting asterisks in the desired print positions. Any Form Line containing one or more numerical fields must be followed by a Content Line which lists the names of the Attributes whose



```
PROGRAMMER ____________                                   PAGE ___ OF ___  FOLD LINE                     PAGE ___ OF ___
PROBLEM ____________        SIMSCRIPT REPORT GENERATOR LAYOUT FORM
DATE ____________                                     LEFT HAND CARDS    RIGHT HAND CARDS

                          PRINT POSITIONS                                   PRINT POSITIONS

  REPØRT ANNUAL

                                 S T Ø C K   S T A T U S   R E P Ø R T

                                                                        AVERAGE
          STØCK       REØRDER     REØRDER    STØCK       DEMAND        SIX MØNTH     UNIT
         NUMBER        PØINT      QUANTITY   ØN HAND   THIS MØNTH      DEMAND        PRICE

         ***A**         ***         ***        ***          **          **.**        *.**
          SN(I),       RP(I),      RQ(I),     STK(I),     DTM(I),      ASMD(I),     PRICE(I)

  FØR I = (I)(NSN), WITH(DTM(I))GR(0)

                        END                                               END
```

```
                          S T O C K   S T A T U S   R E P O R T

                                                                            AVERAGE
   STOCK       REORDER      REORDER      STOCK        DEMAND        SIX MONTH       UNIT
   NUMBER       POINT       QUANTITY    ON HAND     THIS MONTH       DEMAND         PRICE

   4N-118         17          120          84           1              3.81          0.10
   546-NA          9            1           9           1              1.30        485.60
   9-2778          2            3           3           1              0.17          5.92
   421-13        300          140         392         584           2489.75         27.00
   ANM421          1            4           2           1              0.33          3.58
   191-52          0            1           0           1              0.17       9000.00
   1914CN         25          285         137           1              8.24          0.02
   C97-11          2           26          26           2              2.89          1.04
   C97-18         12           12          14           7             17.66         12.75
   420554          1           37          23           2              0.33          0.05
   420555          0           12           7           1              0.17          0.25
   143187          5            7           4           3              2.90         13.80
   2-8238         13          295         119           1              4.04          0.01
```

Fig. 2 — SIMSCRIPT Report Generator Layout Form and Stock Status Report

values are to be printed in the numerical fields. A variety of other control features are also provided in the SIMSCRIPT Report Generator; they are described in detail in Chapter 10.

SUMMARY

In SIMSCRIPT, the Status of a simulated system is described in terms of Entities, Attributes of Entities, and Sets of Entities. Each type of Entity, Attribute, and Set is defined on the SIMSCRIPT Definition Form. .Status is changed at points in simulated time called Events. A separate Event Routine must be written for each different kind of Event to be included in the simulation. For convenience in writing these routines, the SIMSCRIPT source language contains a variety of commands especially adapted to simulation problems. Some of these commands have already been illustrated.

In SIMSCRIPT programs, each kind of Event may occur repeatedly and at any desired point in simulated time. Exogenous Events are initiated from outside the simulation process by means of the Exogenous Event Tape. Endogenous Events are initiated from within the simulation process by CAUSE statements in previously executed subprograms. A Timing routine is automatically provided to keep track of simulated time and call the various Event routines in the proper sequence.

Additional features include subscripted subscripts, "dimension-free" source and object programs, and printed-output routines automatically generated by means of the Report Generator Layout Form.

Chapter 2

VARIABLES AND IDENTIFICATION NUMBERS

SIMSCRIPT contains three different kinds of variables, referred to as (1) Local Variables, (2) Temporary Attributes (Attributes of Temporary Entities and Event Notices), and (3) Permanent Attributes (Attributes of Permanent Entities). This Chapter describes the properties of each, and the use of subscripts or identification numbers.

In the following discussion the term "System Variable" includes Temporary and Permanent Attributes but excludes Local Variables. The term "Variable" includes all three.

LOCAL VARIABLES

Except as noted below, Local Variables must conform to all rules prescribed for FORTRAN variables, including the conventions for naming integer and floating point variables. Local Variables are defined by the source language statements they appear in; no other definition is required. In addition to FORTRAN rules, the name of a SIMSCRIPT Local Variable may not contain two consecutive "X's."

Since Local Variables are not available from one subroutine to another, the same name may be given to different Local Variables in different subroutines without conflict.

Local Variables are rarely subscripted; if they are, they must appear in a DIMENSION statement. As in the case of Temporary and Permanent Attributes described below, a subscript of a Local Variable may be any integer expression. The Variables in this expression may in turn be subscripted to any order.

TEMPORARY ATTRIBUTES

Storage space for Temporary Attributes is assigned each time a Temporary Entity or Event Notice is created during execution of the object program. When the Entity is destroyed, this space is returned for use in storing the values of other Temporary Attributes.

The name of each kind of Temporary Attribute must be defined on the SIMSCRIPT Definition Form. It may consist of one to five letters or numbers, one of which must be a letter. It may not end in "F" or contain "XX" or a special character such as $, +, /, etc. Examples of permissible names are COST, ITEM, XY3, 3A5. Examples of names not permitted are AXX, COEF, A$B, and DEMAND.

A Temporary Attribute is always referred to in the source program by its name plus a single subscript. The subscript identifies the particular Temporary Entity or Event Notice of which the variable is an Attribute. The subscript may be a Local or System Variable and may itself be subscripted without limit, e. g., COST(HOUSE(OWNER(DOG))).

When a new Entity is created by a CREATE statement, space for storing its Attribute values is allocated in the amount specified on the Definition Form. This record is given an identification number, and the Variable specified in the CREATE statement is automatically set equal to this number. *

For example, the statement "CREATE DOG CALLED FIDO" allocates the required amount of storage for the Attributes of a DOG, and sets FIDO equal to the identification number of the Attribute record. DOG is a type of Temporary Entity, and FIDO is a Local Variable. If Entities of the type called DOG have an Attribute called AGE, the AGE of the particular DOG called FIDO would be referred to as AGE (FIDO).

The Variable set equal to the identification number in the CREATE statement may be either a Local Variable, Temporary Attribute, or Permanent Attribute. Like any number, this value may also be given to other additional Local or System Variables. The identification number of a particular Entity may be made available to subsequent Event routines or subroutines by giving its value to a subsequently identifiable Temporary or Permanent Attribute, by use of the various kinds of Set operations, or in case of a subroutine by giving its value to an argument of the CALL and SUBROUTINE statements.

In addition to being used as subscripts for Temporary Attributes, Variables whose values are the identification numbers of Entities also appear in the source language statements, such as CREATE, FILE, REMOVE FIRST, etc., provided for the manipulation of Entities.

The name of an Entity type as specified on the Definition Form may also be used as the name of a Local Variable without conflict. This is a useful feature since it permits statements such as CREATE DOG to be equivalent to CREATE DOG CALLED DOG. In this case, DOG is both the name of a type of Temporary Entity and the name of a Local Variable. The age of the particular dog would be referred to as AGE(DOG) instead of AGE(FIDO) as in the earlier example. Either the "long form" (e. g., CREATE DOG CALLED DOG) or the "short form" (e. g., CREATE DOG) may be used when writing CREATE, DESTROY, CAUSE, or CANCEL statements. Although

*The identification number is equal to one greater than the absolute address of the first word of the Master Record, as discussed in Chapters 13 and 15.

the identification number of a Temporary Entity is an integer, a floating point Local Variable such as DOG or FIDO may be used in the CREATE statement for ease in assigning names, so long as this Variable does not appear in subsequent arithmetic operations. The STORE statement is provided so that the value of an identification number may be given to another Variable without regard to mode.

PERMANENT ATTRIBUTES

If the number of Entities of a particular type cannot change during the simulation run, the Entity type and its Attributes should be defined as "Permanent" rather than "Temporary" on the Definition Form. When appropriate, it is advantageous to use Permanent Attributes because they require less storage space, can be identified by their Entity's ordinal designation (e.g., the "second machine" or the "third man"), and can be controlled as indexed variables without using the more time— and space—consuming Set operations.

Attributes of Permanent Entities are available to all subroutines, and their names are formed according to the same naming rules previously given for Temporary Attributes.

Subscripts of Permanent Attributes, like those of Temporary Attributes, identify the particular Entity which the Attribute describes. However, in the case of Permanent Attributes the value of the subscript is the ordinal designation of the particular Entity. For example, if DEPOT is a type of Permanent Entity, the identification numbers of the individual DEPOTs run from "1" up to the number of DEPOTs. If SIZE is an Attribute of DEPOT, SIZE(1) is the size of the first DEPOT and SIZE(I) is the size of the Ith DEPOT. If BASE is also a type of Permanent Entity, the identification numbers of the individual BASEs run from "1" up to the number of BASEs. The size of the first BASE may be written SIZEB(1); that of the Ith BASE, SIZEB(I). Note that Permanent Entities of two different types may have the same identification number (both the first DEPOT and the first BASE are identified by the number "1"), but they may not have Attributes with exactly the same name.

The subscript of a Permanent Attribute may be an integer Local or System Variable, an integer constant, or an integer expression containing Variables and/or constants. The Variables in a subscript may in turn be subscripted to any order. Unlike the subscripts of Temporary Attributes, those of Permanent Attributes can and often do appear in arithmetic operations; therefore, it is not advisable to use Local Variables with floating point names as subscripts of Permanent Attributes.

The identification numbers of Permanent Entities are made available from one routine to the next by the same procedures mentioned for Temporary Entities. They may also be controlled by means of the FOR and FOR EACH "ENTITY" statements described in Chapter 5. Unlike Temporary Attributes, which always have a single subscript, Permanent Attributes may have no subscript, one, or two.

Unsubscripted Permanent Attributes, sometimes called System Attributes, do not have an Entity which they describe, but in themselves may be thought of as Attributes of the entire simulated system. (The simulated system is conceptually a kind of Permanent Entity, but it is not necessary to define it as such.) System Attributes are stored as single numbers and are referred to by their source language names as defined on the Definition Form.

Two interpretations are possible for double-subscripted Permanent Attributes. They may be thought of as Attributes of pairs of Permanent Entities; for example, the transit time between two different supply depots would be expressed as a double-subscripted Permanent Attribute such as TRANT(I, J). They may also be thought of as Attributes that may have more than one value for each Entity. For example, the number of targets assigned to bomber squadrons may vary from squadron to squadron. The identification of the targets could be treated as a double-subscripted Attribute of the Permanent Entity, "squadron."* In either case, the Attributes are stored as tables. The first case would be a rectangular table each dimension of which would equal the number of a particular type of Permanent Entity. The second case is referred to in subsequent discussion as a "Ragged Table" in which the number of rows is equal to the number of a particular type of Permanent Entity (e.g., "squadron"), and the length of each row varies.

In addition to the Permanent Attributes defined on the Definition Form, the SIMSCRIPT translator automatically defines a number of Permanent Attributes; one most frequently referred to is the current value of simulated time, called TIME. (See Chapter 11 for the complete list of translator-defined System Attributes and functions.)

As with a Temporary Entity, the name of a type of Permanent Entity may be given to Local Variables without conflict, provided the rules for naming Local integer variables are not violated. For example, the name ITEM can be used to identify a type of Permanent Entity, and at the same

*Although Temporary Attributes may have only one value for each Temporary Entity, the same effect may be obtained through the use of a Set. For example, if "squadron" were a type of Temporary Entity, "target" could be defined as another type of Temporary Entity and filed into a Set belonging to a "squadron".

time be used as a Local Variable whose value indicates the number of the particular ITEM involved. As mentioned before, the use of Local Variables with floating point names for this purpose is possible but not advisable in the case of Permanent Entities.

EVENT NOTICE IDENTIFICATION NUMBERS

The handling of Event Notice identification numbers is identical to that of Temporary Entities, with one additonal feature: the identification number is made available to the Event Routine describing the particular kind of Event by means of the CAUSE and ENDOGENOUS EVENT statements. This transfer of the identification number from the causing routine to the Event Routine, as well as the general use of subscripts, is illustrated by the following pair of program fragments which create a job shop work order, assign its next destination, and cause its arrival at that machine group:

`CREATE ORDER`

Equivalent to CREATE ORDER CALLED ORDER. Creates an Attribute record as specified on the Definition Form for an Entity of the type called ORDER. Also sets a Local Variable called ORDER equal to the identification number of this record.

`LET DEST(ORDER)=NEXT`

DEST is an Attribute of the type of Temporary Entity ORDER and is defined as such on the Definition Form. The subscript ORDER is the Local Variable whose value is the Temporary Entity identification number established by the previous statement. NEXT might be a Local or System Variable previously set equal to the identification number of a particular machine group.

`CREATE ARRVL`

Equivalent to CREATE ARRVL CALLED ARRVL. Creates an Attribute record for the Event Notice called ARRVL. Sets a Local Variable called ARRVL equal to its record identification number.

`STORE ORDER IN ORDRA(ARRVL)`

Stores the identification number of the ORDER called ORDER as an Attribute of the ARRVL called ARRVL.

`CAUSE ARRVL AT TIME + TRANT`

Equivalent to CAUSE ARRVL CALLED ARRVL AT TIME + TRANT. Causes the Timing routine to call for the ENDOGENOUS Event ARRVL routine when simulation time is equal to TIME +

TRANT. The value of the Local Variable (in this case ARRVL) will automatically be given to another Local Variable called ARRVL when the ENDOGENOUS EVENT ARRVL routine is called.

When the particular Event scheduled by the preceding statement becomes the most imminent, the Timing Routine will set TIME equal to the time at which the Event is to occur and call the routine called ENDOGENOUS EVENT ARRVL. This routine might begin as follows:

```
ENDOGENOUS EVENT ARRVL
```

Each type of Endogenous Event and its corresponding type of Event Notice must always have the same name. The value of the Local Variable mentioned in the CAUSE statement is automatically given to a Local Variable in the Endogenous Event Routine when the Event Routine is called. This latter Local Variable also always has the same name as the Endogenous Event and its Event Notice (there is no "long form" of the ENDOGENOUS EVENT statement). Note that the same name may appear repeatedly as the names of Local Variables in various routines, as the name of a type of Event Notice, and as the name of an Endogenous Event Routine.

```
STORE DEST(ORDRA(ARRVL)) IN MG
```

Sets the Local Variable MG equal to the identification number previously obtained from the variable NEXT.

USE OF IDENTIFICATION NUMBERS IN SET OPERATIONS

Sets of Entities are in fact collections of Entity identification numbers. These numbers represent the members of the Set and are arranged on a "last–in–first–out" (LIFO) or "first–in–first–out" (FIFO) basis, or they are "ranked" according to the value of one of the members' Attributes. The type of Set organization (LIFO, FIFO, or Ranked) is specified on the Definition form.

Identification numbers are inserted into Sets by means of the FILE statement. They may be removed from Sets by the REMOVE FIRST or REMOVE "SPECIFIC" statements. If there is a Set of DOGs called CAGE, the statement FILE FIDO IN CAGE inserts the value of FIDO into the collection of identification numbers identifying the various DOGs in the Set.

When this particular identification number becomes first in the Set, the next—executed REMOVE FIRST statement will remove this identification number from the Set and assign its value to whatever Variable is mentioned in the REMOVE FIRST statement. For example, if the identification number previously referred to as FIDO were next to be removed from the Set, the statement REMOVE FIRST HOUND FROM CAGE would delete it from the Set and assign its value to the Local Variable HOUND. These statements could also have been written FILE DOG IN CAGE and REMOVE FIRST DOG FROM CAGE, using "DOG" as a Local Variable.

A Set may have zero, one or two subscripts. Unsubscripted Sets are considered as belonging to the entire system, single—subscripted Sets belong to individual Entities, and double—subscripted Sets belong to pairs of Entities. The subscripts are the identification numbers of the owning Entities.

For the translator to accomplish the various Set operations, it is necessary to define one or more special Attributes for the type of Entity belonging to the Set and for the type of Entity owning the Set.

The member type of Entity must have an Attribute whose value is the identification number of the Entity immediately succeeding it in the set ("succeeding" in terms of the order in which it will be removed). If the Set is Ranked, the member type of Entity must also have an Attribute whose value is the identification number of the preceding member of the Set.

The owner type of Entity for each Set must have an Attribute whose value is the identification number of the first member to be removed from the Set. If the set has a FIFO or Ranked organization, the owner type of Entity must also have an Attribute whose value is the identification number of the last member to be removed from the Set.

In the definition of these Attributes on the Definition Form, their names are always formed by prefixing a single letter to the name of the Set. The prefixes corresponding to "first," "last," "predecessor," and "successor" are "F," "L," "P," and "S," respectively. For example, if DOGs are members of the Set called CAGE, DOG must have an Attribute called "SCAGE." If CAGE is a Ranked Set, DOG must also have an Attribute called "PCAGE." Assuming CAGE is unsubscripted and is therefore owned by the system, there must be an unsubscripted Permanent Attribute called "FCAGE." If the Set is FIFO or Ranked, there must also be an unsubscripted Permanent Attribute called "LCAGE."

The meaning of each type of special Attribute, the manner in which its name is formed, and the types of Set organization that require it are summarized below:

Meaning	Attribute Name	Requiring Types of Set Organization	Describes Member or Owner Entity
Predecessor in Set	P"Set name"	Ranked	Member
Successor in Set	S"Set name"	All	Member
First in Set	F"Set name"	All	Owner
Last in Set	L"Set name"	Ranked, FIFO	Owner

The values of the "first," "last," "predecessor," and "successor" Attributes are automatically changed as required by FILE, REMOVE FIRST, and REMOVE "SPECIFIC" statements. These Attributes are available to all subprograms, but their values may not be modified by source statements other than FILE, REMOVE FIRST, and REMOVE "SPECIFIC."

Note that because of the required owner Attributes, double-subscripted Sets may belong only to Permanent Entities, since double-subscripted Attributes are not permitted for Temporary Entities.

CONSTANTS AND EXPRESSIONS

Constants and expressions may be of integer or floating point mode. Values of integer constants are written with no decimal point. Values of floating point constants are written with a decimal point (either preceding, following, or between the digits).

An integer expression consists of one or more integer Variables and/or integer constants combined by arithmetic operations. Similarly, a floating point expression consists of one or more floating point Variables and/or floating point constants combined by arithmetic operations. Mixed expressions are not permitted; however, integer expressions may appear as subscripts in floating point expressions. Expressions may contain both System and Local Variables.

The notation and conventions for arithmetic operations and for the use of parentheses are the same in SIMSCRIPT as they are in FORTRAN.

Chapter 3

AN EXAMPLE OF A SIMSCRIPT SOURCE PROGRAM

To illustrate the writing of a SIMSCRIPT source program, suppose a simple job shop simulation were to be programmed with the following features. The simulated shop consists of a number of machine groups. The machines in a group are considered identical for all practical purposes. The number of groups and the number of machines within a group are specified separately for each simulation run and remain constant throughout the run. Orders come into the shop from outside the simulation process at arbitrary points in time. Each order must go through operations at each of a series of different machine groups. The routing through the shop and the processing time at each machine group are specified for each order and may vary for different orders. After an order is processed by one machine group, it is immediately routed to the next machine group. It is put into process at once if there is a machine available; otherwise it is put into the queue of orders waiting for that machine group. Orders enter and leave the queues on a first–in–first–out basis.

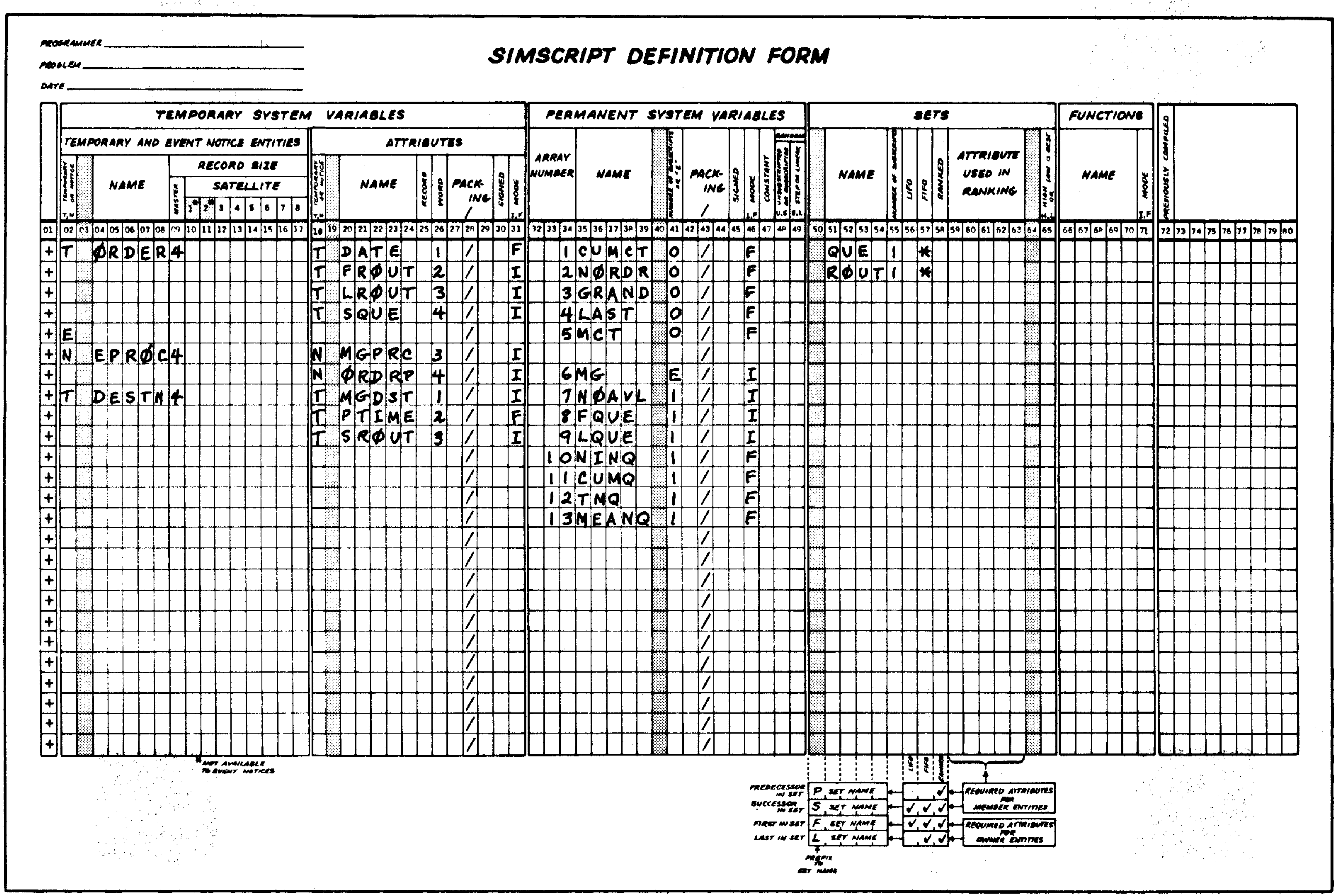
The results desired from the simulation are the average length of time that orders remain in the shop, and the average number of orders in the queue of each machine group. These results are to be obtainable at arbitrarily specified dates during the simulation run.

The following discussion describes one possible way this simulation could be programmed in SIMSCRIPT.

The Entity types used to define the shop Status include the Permanent Entity MG ("machine group"), the Temporary Entity ORDER, and the Event Notice EPROC for an Endogenous Event describing the end of a process.

The queue of orders waiting for a machine group is treated as a "first–in–first–out" Set called QUE(MG) having the type of Entity called ORDER as members and belonging to the type of Entity called MG. This FIFO Set requires that Attributes called FQUE (first in queue) and LQUE (last in queue) be defined for the Entity MG. It also requires that an Attribute called SQUE (successor in queue) be defined for the type of Entity called ORDER.

In describing the routing of an order, it is convenient to treat the routing as a Set of "destinations" called ROUT(ORDER). This approach requires an additional type of Entity called DESTN. Attributes called FROUT and LROUT (first and last destinations in routing) must be defined for the Entity ORDER, and the Attribute SROUT (successor in routing) must be defined for the Entity DESTN.



PROGRAMMER ____________________

PROBLEM ____________________

DATE ____________________

SIMSCRIPT DEFINITION FORM

	TEMPORARY SYSTEM VARIABLES: TEMPORARY AND EVENT NOTICE ENTITIES		RECORD SIZE		ATTRIBUTES							PERMANENT SYSTEM VARIABLES							RANDOM		SETS							FUNCTIONS			
	TEMPORARY OR NOTICE T,N	NAME	MASTER	SATELLITE 1* 2* 3 4 5 6 7 8	TEMPORARY OR NOTICE T,N	NAME	RECORD	WORD	PACKING	SIGNED	MODE I,F	ARRAY NUMBER	NAME	NUMBER OF SUBSCRIPTS OR "E"	PACKING	SIGNED	MODE I,F	CONSTANT	UNSUBSCRIPTED OR SUBSCRIPTED U,S	STEP OR LINEAR S,L	NAME	NUMBER OF SUBSCRIPTS	LIFO	FIFO	RANKED	ATTRIBUTE USED IN RANKING	HIGH OR LOW IS BEST H,L	NAME	MODE I,F	PREVIOUSLY COMPILED	
01	02 03	04 05 06 07 08	09	10 11 12 13 14 15 16 17	18 19	20 21 22 23 24	25	26	27 28 29	30	31	32 33 34	35 36 37 38 39	40 41	42 43 44	45	46	47	48	49	50 51 52 53 54	55	56	57	58	59 60 61 62 63	64 65	66 67 68 69 70	71	72	73 74 75 76 77 78 79 80
+	T	ØRDER	4		T	DATE		1	/		F	1	CUMCT	0	/		F				QUE	1		*							
+					T	FRØUT		2	/		I	2	NØRDR	0	/		F				RØUT	1		*							
+					T	LRØUT		3	/		I	3	GRAND	0	/		F														
+					T	SQUE		4	/		I	4	LAST	0	/		F														
+	E								/			5	MCT	0	/		F														
+	N	EPRØC	4		N	MGPRC		3	/		I				/																
+					N	ØRDRP		4	/		I	6	MG	E	/		I														
+	T	DESTN	4		T	MGDST		1	/		I	7	NØAVL	1	/		I														
+					T	PTIME		2	/		F	8	FQUE	1	/		I														
+					T	SRØUT		3	/		I	9	LQUE	1	/		I														
+									/			10	NINQ	1	/		F														
+									/			11	CUMQ	1	/		F														
+									/			12	TNQ	1	/		F														
+									/			13	MEANQ	1	/		F														
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																
+									/						/																

* NOT AVAILABLE TO EVENT NOTICES

	PREFIX TO SET NAME	LIFO	FIFO	RANKED	
PREDECESSOR IN SET	P SET NAME			✓	REQUIRED ATTRIBUTES FOR MEMBER ENTITIES
SUCCESSOR IN SET	S SET NAME	✓	✓	✓	
FIRST IN SET	F SET NAME	✓	✓	✓	REQUIRED ATTRIBUTES FOR OWNER ENTITIES
LAST IN SET	L SET NAME		✓	✓	

Fig. 3 — Definitions for Example Program

Among the other Attributes of the various Entities are the number of machines available in a machine group, NOAVL(MG), the date of receipt of an order, DATE(ORDER), and the processing time at a destination, PTIME(DESTN). These Entities, Attributes, and Sets, and additional Attributes for storing identification numbers and interim result totals, are defined on the Definition Form as shown in Fig. 3.

In the present example, the shop is simulated with two different kinds of Events: EXOGENOUS EVENT ORDRIN representing the receipt of an order in the shop, and ENDOGENOUS EVENT EPROC representing the end of the processing of an order at a machine group. Two additional kinds of Events, EXOGENOUS EVENT ANALYZ and EXOGENOUS EVENT ENDSIM, are used to analyze the results and end the simulation. In the present example two Subroutines and one Report Routine are also used.

In order that the SIMSCRIPT translator may generate an appropriate Timing Routine, each type of Event is enumerated in an "Events List" as illustrated in Fig. 4. The conventions for preparing an Events List are described in Chapter 9. In addition to the Events List, one Event Routine is written to describe each type of Event.

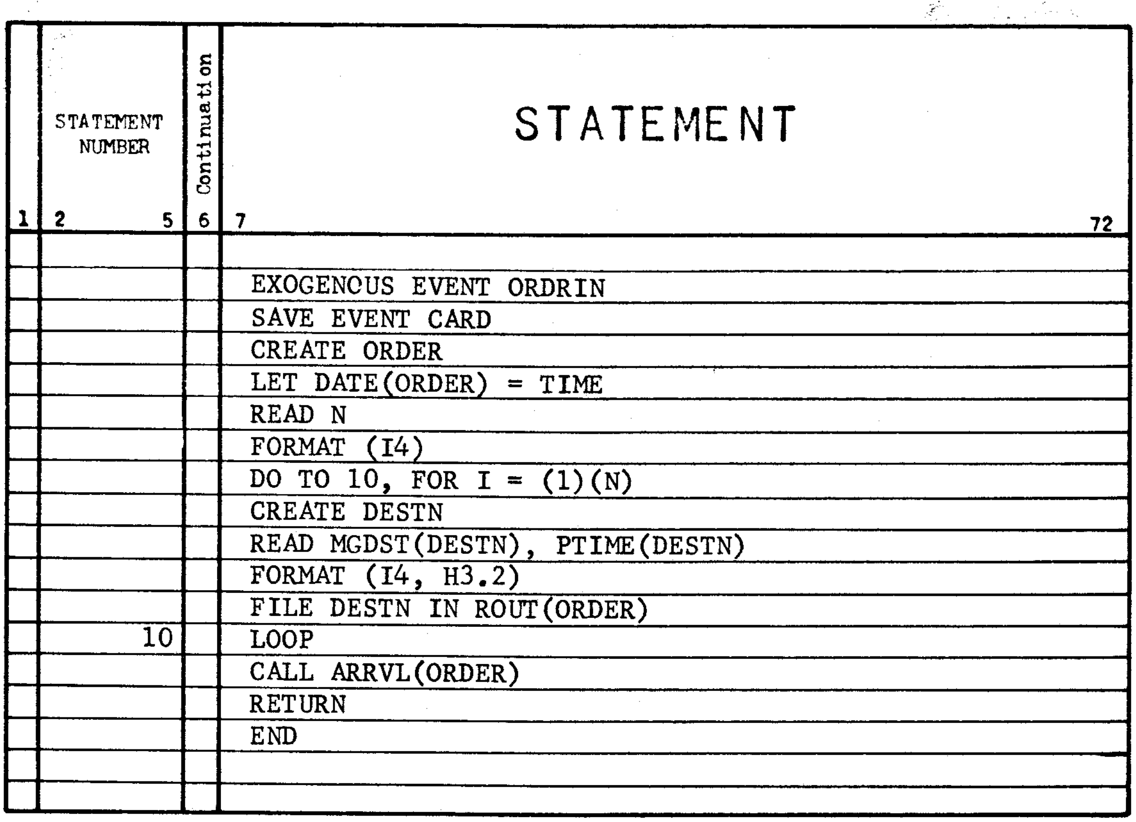
1	STATEMENT NUMBER 2 5	Continuation 6	STATEMENT 7 72

```
      EVENTS

         3  EXOGENOUS
               ORDRIN (1)
               ANALYZ (2)
               ENDSIM (3)

         1  ENDOGENOUS
               EPROC


      END
```

Fig. 4 — Events List for Example — Job Shop Simulation

EXOGENOUS EVENT ORDRIN

Fig. 5 shows the required subroutine describing the changes in Status when an order is received in the shop.



STATEMENT NUMBER
Continuation
STATEMENT
1
2
5
6
7
72

```
      EXOGENCUS EVENT ORDRIN
      SAVE EVENT CARD
      CREATE ORDER
      LET DATE(ORDER) = TIME
      READ N
      FORMAT (I4)
      DO TO 10, FOR I = (1)(N)
      CREATE DESTN
      READ MGDST(DESTN), PTIME(DESTN)
      FORMAT (I4, H3.2)
      FILE DESTN IN ROUT(ORDER)
   10 LOOP
      CALL ARRVL(ORDER)
      RETURN
      END
```

Fig. 5 — Exogenous Event Routine Describing the Receipt of an Order

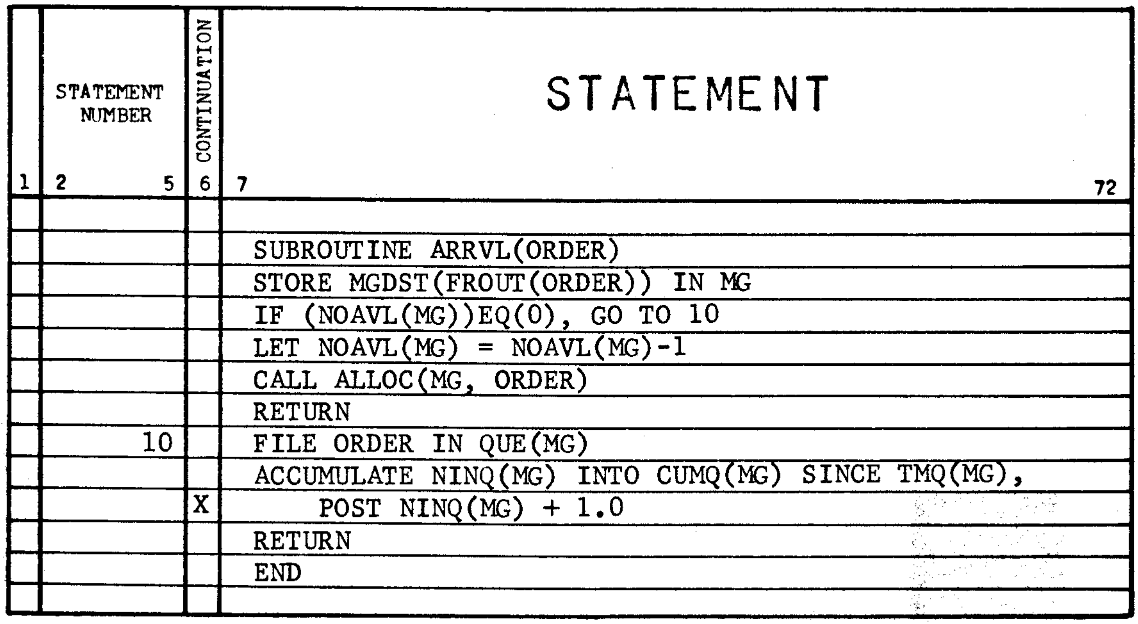
In general terms, the subroutine shown in Fig. 5 creates a record in memory for storing the Attribute values of the new order, notes the date of receipt as one of its Attributes, and reads in the number of destinations in the routing of the order. For each destination, it creates a record for storing the destination's Attribute values and reads and stores two of these values, namely, the processing time and the machine group involved. The destination is then filed into its proper place in the routing of the order. Finally, the order is moved to its first destination by calling a subroutine describing the arrival of an order at its next destination. The specific meaning of each of the commands is described on the following page.

EXOGENOUS EVENT ORDRIN	Indicates an Event Routine describing an Exogenous Event called ORDRIN.
SAVE EVENT CARD	Indicates that additional data are to be read from the Event Card.
CREATE ORDER	Creates a record in memory for storing the Attributes of the Entity type ORDER. The configuration of the record is specified when defining ORDER on the Definition Form. The identification number of this record is given to a Local Variable also called ORDER. In other words, the statement is equivalent to CREATE ORDER CALLED ORDER.
LET DATE(ORDER) = TIME	Sets the date of receipt of the order equal to the current value of simulated time. DATE is an Attribute of the Entity type called ORDER. DATE(ORDER) is an Attribute of the particular ORDER whose identification number is equal to the Local Variable ORDER. TIME is a System Attribute that is automatically defined by the translator and is always available. Its value is set automatically equal to current simulated time prior to the calling of each Event Routine.
READ N FORMAT (I4)	Reads the integer N from the first four data columns of the Event Card. N is equal to the number of destinations in the routing of the order. The routing is described by N data cards following the Event Card on the Exogenous Event Tape.
DO TO 10, FOR I = (1)(N)	Indicates that the following sequence of statements through statement 10 is to be excuted N times.
CREATE DESTN	Creates a record in memory for storing the Attributes of a particular destination of the order.
READ MGDST(DESTN), PTIME(DESTN) FORMAT (I4, H3.2)	Reads and stores two of the destination's Attributes, namely, the machine group involved MGDST(DESTN) and processing time required at that destination PTIME(DESTN).

	FILE DESTN IN ROUT(ORDER)	Files the destination into the routing of the order on a first-in-first-out basis as specified in defining ROUT on the Definition Form.
10	LOOP	Returns control to the DO TO statement.
	CALL ARRVL(ORDER)	Calls on the ARRVL Subroutine describing the arrival of an order at a machine group. The argument ORDER indicates the identification number of the particular order involved.
	RETURN	Returns control to the Timing routine.
	END	Designates the physical end of the subprogram.

SUBROUTINE ARRVL

Figure 6 shows the Subroutine describing the arrival of an order at a machine group. If a machine is available, the "number of machines available" is reduced by one and a Subroutine is called which allocates a machine to an order. If a machine is not available, the order is placed in the queue of the machine group on a first–in–first–out basis, and data related to the average size of the queue are accumulated.



1	STATEMENT NUMBER 2 5	CONTINUATION 6	STATEMENT 7 72
			SUBROUTINE ARRVL(ORDER)
			STORE MGDST(FROUT(ORDER)) IN MG
			IF (NOAVL(MG))EQ(0), GO TO 10
			LET NOAVL(MG) = NOAVL(MG)-1
			CALL ALLOC(MG, ORDER)
			RETURN
	10		FILE ORDER IN QUE(MG)
			ACCUMULATE NINQ(MG) INTO CUMQ(MG) SINCE TMQ(MG),
		X	POST NINQ(MG) + 1.0
			RETURN
			END

Fig. 6 – Subroutine Describing the Arrival of an Order at a Machine Group

The specific meaning of each of the statements is described below.

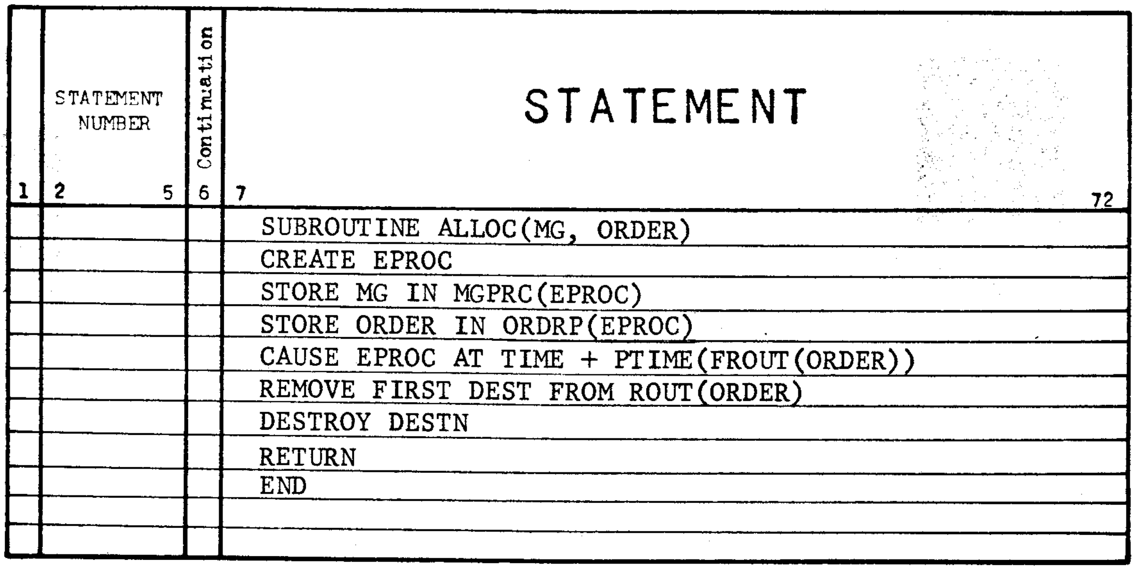
`SUBROUTINE ARRVL(ORDER)`	Indicates a Subroutine called ARRVL with one argument. The Local Variable ORDER identifies the order involved.
`STORE MGDST(FROUT(ORDER)) IN MG`	Stores the identification number of the machine group destination that is currently first in the routing of the order in the Local Variable MG.
`IF (NOAVL(MG))EQ(0), GO TO 10`	Transfers control to statement number 10 if number of machines available equals zero. Otherwise, execution continues with the following statement.
`LET NOAVL(MG)=NOAVL(MG)-1`	If a machine is available at the machine group, the number available is decreased by one.
`CALL ALLOC(MG,ORDER)`	Calls the Subroutine which allocates a machine to an order. The arguments given to the Subroutine indicate the machine group and the order involved.
`RETURN`	Returns control to the Timing routine.
`10 FILE ORDER IN QUE(MG)`	If a machine is not available, the order is filed into the queue of the machine group.

`ACCUMULATE NINQ(MG) INTO`
`CUMQ(MG) SINCE TMQ(MG),`
`POST NINQ(MG) + 1.0`
	Accumulates the data needed for computing the average size of the machine group's queue. It multiplies the number of orders in the queue by the elapsed time since the previous change in the queue and adds this to the cumulative total. The number in the queue is then increased by one.

`RETURN`	Returns control to the Timing routine.
`END`	Designates physical end of subprogram.

SUBROUTINE ALLOC

To allocate a machine to an order, the Subroutine shown in Fig. 7 creates an Event Notice for the coming "end of process" and posts the values of two Attributes to this record. The end of process is then scheduled for occurrence following the lapse of the particular processing time. The present destination is removed from the routing of the order and destroyed.



STATEMENT NUMBER | Continuation | STATEMENT

1 2 5 6 7 72

```
SUBROUTINE ALLOC(MG, ORDER)
CREATE EPROC
STORE MG IN MGPRC(EPROC)
STORE ORDER IN ORDRP(EPROC)
CAUSE EPROC AT TIME + PTIME(FROUT(ORDER))
REMOVE FIRST DEST FROM ROUT(ORDER)
DESTROY DESTN
RETURN
END
```

Fig. 7 – Subroutine for Allocating a Machine to an Order

This subroutine is described in detail below.

SUBROUTINE ALLOC(MG,ORDER)

Indicates a subroutine called ALLOC with two arguments supplied by the calling routine. The first argument is the identification of the machine group involved; the second argument identifies the order.

CREATE EPROC

Creates a record for storing the Attribute values of the end-of-process Event Notice.

STORE MG IN MGPRC(EPROC)

Sets the Attribute MGPRC of the Event Notice EPROC equal to the machine group specified by the calling routine.

STORE ORDER IN ORDRP(EPROC)

Sets the Attribute ORDRP of the Event Notice EPROC equal to the order specified by the calling routine.

CAUSE EPROC AT TIME + PTIME(FROUT(ORDER))

Instructs the Timing routine to call the ENDOGENOUS EVENT EPROC routine when simulated time is equal to current time plus the processing time for the destination that is first in the routing of the order.

REMOVE FIRST DESTN FROM ROUT(ORDER)

Removes the first destination from the routing of the order.

DESTROY DESTN	Sets the contents of the destination record equal to zero and makes this storage space available to subsequent CREATE statements.
RETURN	Returns control to the calling routine.
END	Designates physical end of the subprogram.

ENDOGENOUS EVENT EPROC

The Event routine describing the end of process for an order at a machine group is shown in Fig. 8. If the order has another destination in the shop, the ARRVL(ORDER) Subroutine is called. If the order has no

1	2 STATEMENT NUMBER 5	6 Continuation	7 STATEMENT 72
			ENDOGENOUS EVENT EPROC
			STORE ORDRP(EPROC) IN ORDER
			STORE MGPRC(EPROC) IN MG
			DESTROY EPROC
C			- DISPOSITION OF THE ORDER -
			IF ROUT(ORDER) IS EMPTY, GO TO 10
			CALL ARRVL(ORDER)
			GO TO 20
	10		LET CUMCT + TIME - DATE(ORDER)
			LET NORDR = NORDR + 1.0
			DESTROY ORDER
C			- DISPOSITION OF THE MACHINE -
	20		IF QUE(MG) IS EMPTY, GO TO 30
			REMOVE FIRST ORDER FROM QUE(MG)
			CALL ALLOC(MG, ORDER)
			ACCUMULATE NINQ(MG) INTO CUMQ(MG) SINCE TMQ(MG),
		X	POST NINQ(MG) - 1.0
			RETURN
	30		LET NOAVL(MG) = NOAVL(MG) + 1
			RETURN
			END

Fig. 8 — Endogenous Event Routine Describing the End Process for an Order at a Machine Group

other destinations, its time in the shop is added to cumulative cycle time for the shop, it is tallied to the number of orders completed, and its record is destroyed. Following the disposition of the order, the routine next deals with the machine that has just become available. If other orders are in the machine group's queue, the first order is removed and given to the Subroutine for allocating a machine to an order, and the data relating to the average size of the queue are then updated. If there is no queue, the number of machines available is increased by one. A detailed description of the Event Routine is given below.

`   ENDOGENOUS EVENT EPROC`	Indicates an Event Routine describing an Endogenous Event called EPROC.
`   STORE ORDRP(EPROC) IN ORDER`	Sets the Local Variable ORDER equal to the identification of the order that has finished processing.
`   STORE MGPRC(EPROC) IN MG`	Sets the Local Variable MG equal to the machine group that has just completed the process.
`   DESTROY EPROC`	Sets the contents of the end-of-process record equal to zero and makes this storage space available to subsequent CREATE statements.
`C  -DISPOSITION OF THE ORDER -`	A Comment. Does not affect the object program.
`   IF ROUT(ORDER) IS EMPTY, GO TO 10`	Tests to see if any destinations remain in the routing of the order.
`   CALL ARRVL(ORDER)`	Calls the previously described Subroutine describing the arrival of an order at a machine group.
`   GO TO 20`	Transfers control to statement number 20.
`10 LET CUMCT=CUMCT+TIME-DATE(ORDER)`	Adds the elapsed time the order has been in the shop to the cumulative cycle time of all previously completed orders.
`   LET NORDR=NORDR+1.0`	Increases the number of orders completed by one.
`   DESTROY ORDER`	Sets the record of the order to zero and makes the space available for the use of subsequent CREATE statements.
`C  -DISPOSITION OF THE MACHINE -`	A comment.

20	`IF QUE(MG) IS EMPTY, GO TO 30`	Tests to see if there is another order in the queue of the machine group that just completed the process.
	`REMOVE FIRST ORDER FROM QUE(MG)`	Removes the order that is first in the queue of the machine group. The Local Variable ORDER now identifies the order just removed from the queue.
	`CALL ALLOC(MG,ORDER)`	Calls the previously described Subroutine that allocates a machine to an order.

	`ACCUMULATE NINQ(MG) INTO CUMQ(MG)`
	`SINCE TMQ(MG), POST NINQ(MG)-1.0`
		Updates the cumulative number in queue and decreases the number in queue by one.

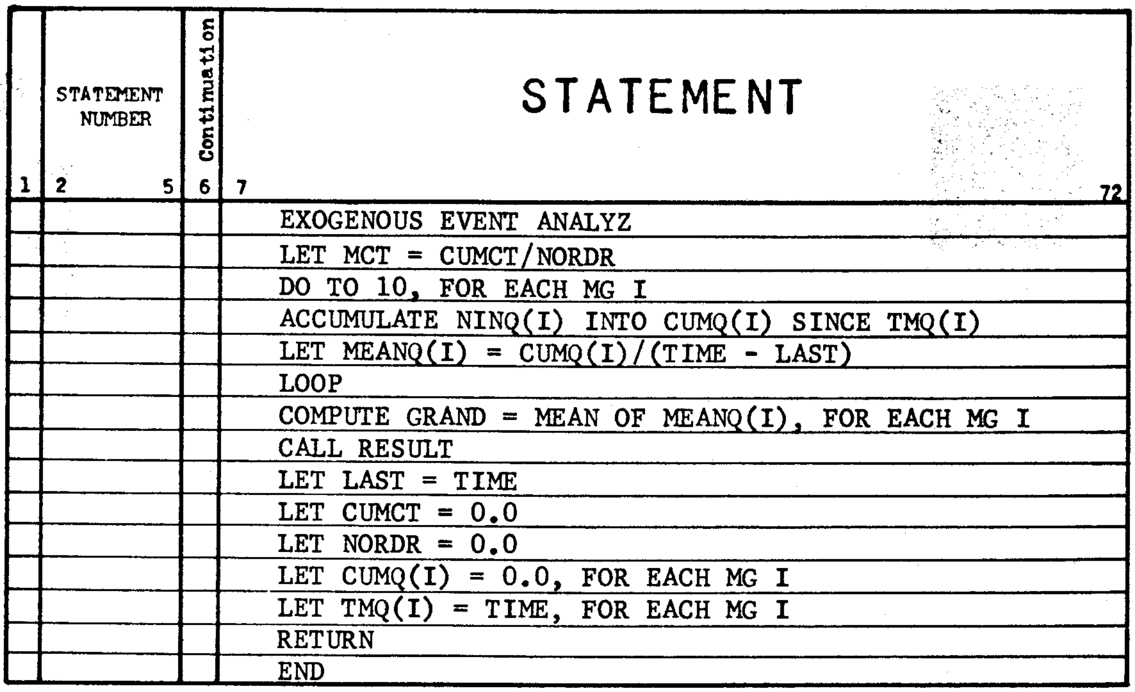
	`RETURN`	Returns control to the Timing routine.
30	`LET NOAVL(MG)=NOAVL(MG)+1`	Increases the number of machines available by one if the queue is empty.
	`RETURN`	Returns control to the Timing routine.
	`END`	Indicates the physical end of subprogram.

The preceding two Event Routines, plus the Subroutines for the arrival of an order at a machine group and for allocating a machine to an order, describe the running of the shop. The remaining routines analyze and report results and stop the simulation run.

EXOGENOUS EVENT ANALYZ

The Exogenous Event routine shown in Fig. 9 analyzes the queue–length and cycle–time data that have accumulated since the previous Report. The various cumulative quantities and time variables are appropriately reset so that the data for the next report can be accumulated. Except for COMPUTE, the meanings of the statements have already been illustrated. The statement COMPUTE GRAND = MEAN OF MEANQ(I), FOR EACH MG I computes the arithmetic mean of the quantities MEANQ(I) and sets GRAND equal to this value.

The EXOGENOUS EVENT ANALYZ routine calls a report routine called RESULT which is written on the Report Generator Layout Form as shown in Fig. 10.



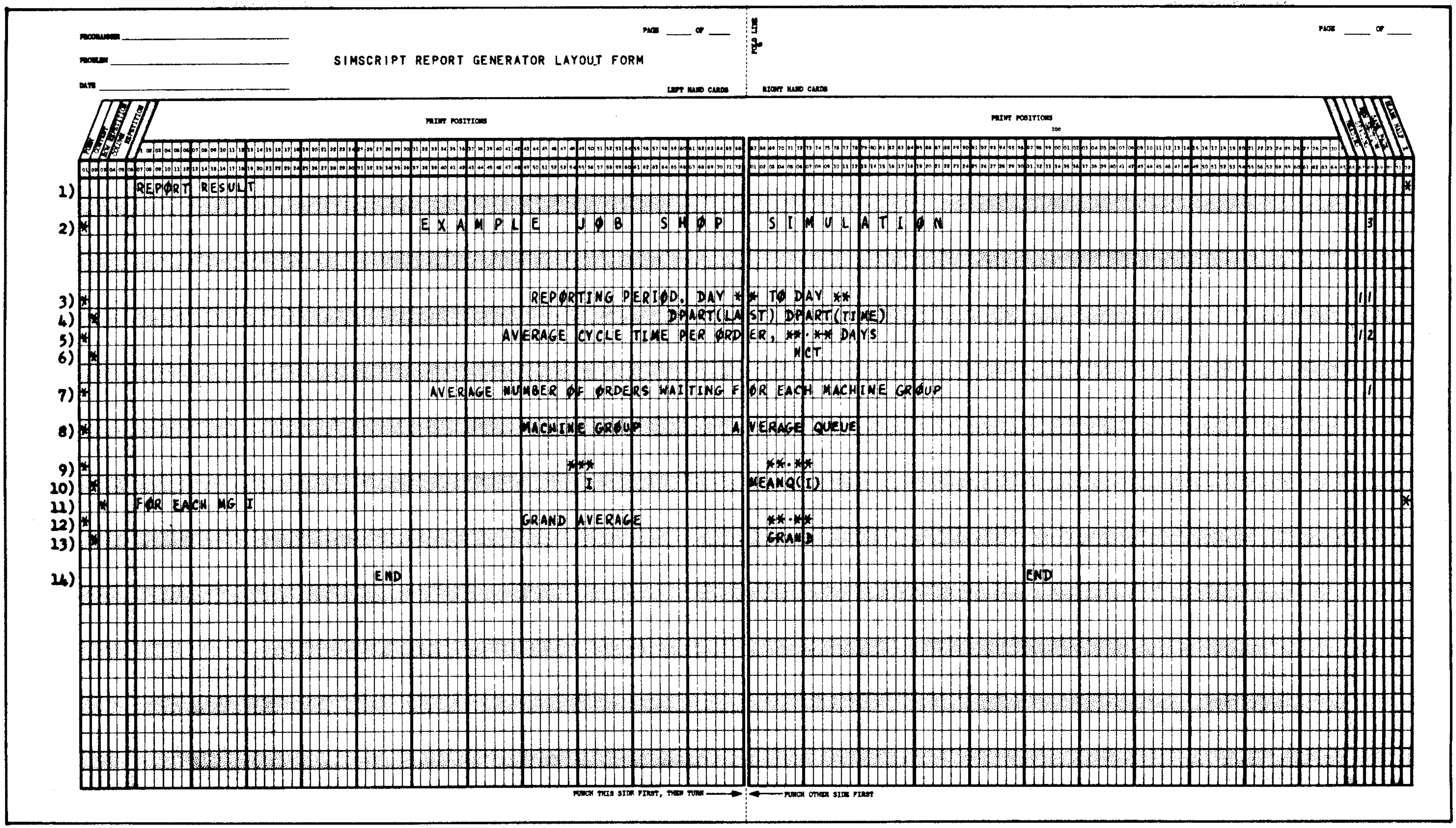
1	STATEMENT NUMBER 2 5	Continuation 6	STATEMENT 7 72
			EXOGENOUS EVENT ANALYZ
			LET MCT = CUMCT/NORDR
			DO TO 10, FOR EACH MG I
			ACCUMULATE NINQ(I) INTO CUMQ(I) SINCE TMQ(I)
			LET MEANQ(I) = CUMQ(I)/(TIME - LAST)
			LOOP
			COMPUTE GRAND = MEAN OF MEANQ(I), FOR EACH MG I
			CALL RESULT
			LET LAST = TIME
			LET CUMCT = 0.0
			LET NORDR = 0.0
			LET CUMQ(I) = 0.0, FOR EACH MG I
			LET TMQ(I) = TIME, FOR EACH MG I
			RETURN
			END

Fig. 9 – Exogenous Event Routine for Analyzing Accumulated Results

REPORT RESULT

For purposes of explanation, each line has been given a number in the left–hand margin of Fig. 10. The meaning of each line is as follows:

1. Indicates the name of the particular Report Routine. Does not appear in the printed output.
2. A "Form Line" that displays text to be printed exactly as indicated. The "3" under "Spacing" specifies that three lines are to be skipped following the printing of this line (the use of this and other control columns is explained in Chapter 10).
3. Except for the asterisks, all characters in this "Form Line" will be printed in the print positions shown. The asterisks designate integer fields for printing the values of the two variables specified in the Content Line immediately following it.
4. A "Content Line" indicating that the "days part" of time variable LAST(time of last report) and TIME(time of this report) are to be printed in the previously specified integer fields. Variables in a Content Line correspond in order to the asterisk fields in the preceding Form Line. They need not be written directly under their respective print positions as they are in Fig. 10.



PROGRAMMER ______________________

PROBLEM ______________________

DATE ______________________

SIMSCRIPT REPORT GENERATOR LAYOUT FORM

PAGE ____ OF ____ LEFT HAND CARDS

FOLD LINE

RIGHT HAND CARDS PAGE ____ OF ____

PRINT POSITIONS

PRINT POSITIONS 100

```
 1)      REPØRT RESULT                                                                                   *
 2) *                           E X A M P L E   J Ø B   S H Ø P   S I M U L A T I Ø N               3
 3) *                                       REPØRTING PERIØD, DAY ** TØ DAY **                     1 1
 4)  *                                                DPART(LAST) DPART(TIME)
 5) *                                     AVERAGE CYCLE TIME PER ØRDER, **.** DAYS                 1 2
 6)  *                                                              MCT
 7) *                              AVERAGE NUMBER ØF ØRDERS WAITING FØR EACH MACHINE GRØUP           1
 8) *                                       MACHINE GRØUP             AVERAGE QUEUE
 9) *                                          ***                      **.**
10)  *                                           I                     MEANQ(I)
11)   *  FØR EACH MG I                                                                               *
12) *                                       GRAND AVERAGE               **.**
13)  *                                                                  GRAND
14)                          END                                                END
```

PUNCH THIS SIDE FIRST, THEN TURN ——→ ←—— PUNCH OTHER SIDE FIRST

Fig. 10 — SIMSCRIPT Report Generator Layout Form (reduced)

5. Another Form Line containing both text to be printed as written and a decimal field defined by asterisks. The decimal point will appear in the print position indicated.

6–10. Additional Form and Content Lines.

11. A "Row Repetition" line specifying that the output defined by the previous pair of Form and Content Lines is to be repeated for each machine group (FOR EACH MG I).

12,13. Form and Content Lines for printing the average queue for all machine groups.

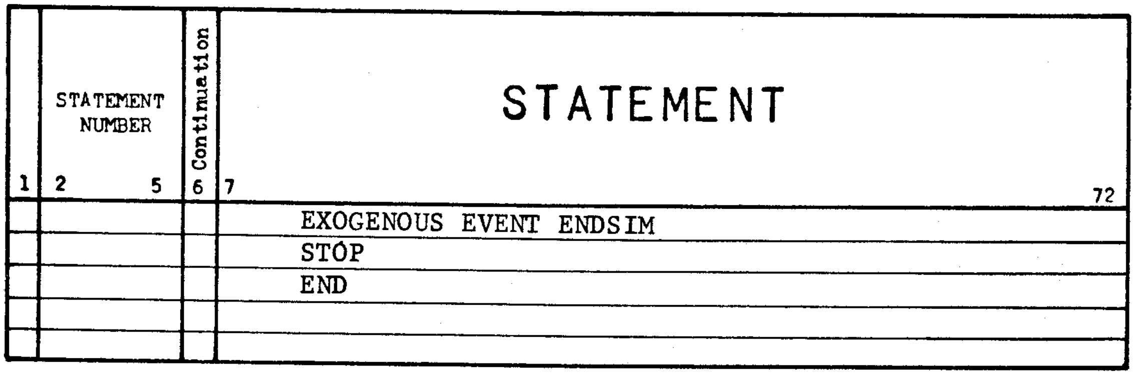
14. Signifies the end of the report.

INITIAL CONDITIONS AND END OF SIMULATION

In order that SIMSCRIPT source and object programs may be "dimension-free," the initial Attribute values for Permanent Entities and for the System are input at the time of object program execution as described in Chapter 14.

If it is desired to start the simulation with Temporary Entities in existence and with Endogenous Events already scheduled, these conditions may be established by a special Exogenous Event routine written for this purpose and occurring at time zero. In the present example, the simulation is started with an empty shop. Orders can be introduced by means of the EXOGENOUS EVENT ORDRIN routine.

Program execution is terminated by the STOP statement. In the present example, the end of simulation might be accomplished by the following Exogenous Event:


STATEMENT NUMBER
Continuation
STATEMENT
1
2
5
6
7
72

```
EXOGENOUS EVENT ENDSIM
STOP
END
```

Fig. 11 – Exogenous Event Routine to Stop the Simulation

Chapter 4

ENTITY OPERATIONS

The following SIMSCRIPT statements are provided for creating and destroying Entities, filing and removing them from Sets, and causing and cancelling Events:

CREATE
DESTROY
CAUSE
CANCEL

FILE
REMOVE FIRST
REMOVE "SPECIFIC"

CREATE STATEMENTS

When a new Temporary Entity or Event Notice first comes into being, a record for storing its Attribute values must be established by means of a CREATE statement.

GENERAL FORM	EXAMPLES

CREATE "Temporary Entity or Event Notice" CALLED "Variable"	CREATE ORDER CALLED I
	CREATE EPROC CALLED E

The CREATE statement accomplishes two actions: first, storage space is assigned for the record of the newly created Temporary Entity or Event Notice; second, the "Variable" mentioned in the CREATE statement is set equal to the identification number of this record. All new records contain zeros. The record configuration for each kind of Temporary Entity or Event Notice is specified on the Definition Form. For convenience, the following short form of the CREATE statement is provided:

GENERAL FORM	EXAMPLES

CREATE "Temporary Entity or Event Notice"
This short form is equivalent to:
CREATE "Temporary Entity or Event Notice" CALLED "Local Variable with same name as the Temporary Entity or Event Notice"
	CREATE ORDER
	which means:
	CREATE ORDER CALLED ORDER

DESTROY STATEMENTS

When a Temporary Entity or Event Notice goes out of existence, a DESTROY statement destroys its record to make this storage space available for the records of new Entities.

GENERAL FORM	EXAMPLES
DESTROY "Temporary Entity or Event Notice" CALLED "Variable"	DESTROY ORDER CALLED I

The "Temporary Entity or Event Notice" mentioned in the DESTROY statement identifies the type of record to be destroyed. The "Variable" indicates the identification number of the record. As with the CREATE statement, a short form of the DESTROY statement is provided:

GENERAL FORM	EXAMPLES
DESTROY "Temporary Entity or Event Notice" This short form is equivalent to: DESTROY "Temporary Entity or Event Notice" CALLED "Local Variable with same name as the Temporary Entity or Event Notice"	DESTROY TRNST which means: DESTROY TRNST CALLED TRNST

CAUSE STATEMENTS

GENERAL FORM	EXAMPLES
CAUSE "Event Notice" CALLED "Variable" AT "floating point expression specifying time that the Event is to occur"	CAUSE PROC CALLED LPR AT TIME + LT

The CAUSE statement instructs the object program to call for an Event Routine called "ENDOGENOUS EVENT 'Event Notice'" at the point in simulated time specified by the floating point expression. The "Variable" must equal the identification number of the Event Notice record. When the

Event Routine is called, the value of this identification number is automatically given to a Local Variable with the same name as the Event and the Event Notice. The CAUSE statement may be written in the short form shown below.

GENERAL FORM	EXAMPLES
CAUSE "Event Notice" AT "time expression" This short form is equivalent to: CAUSE "Event Notice" CALLED "Local Variable with same name as the Event Notice" AT "time expression"	CAUSE ARRVL AT TIME + TRANT which means: CAUSE ARRVL CALLED ARRVL AT TIME + TRANT

CANCEL STATEMENTS

GENERAL FORM	EXAMPLES
CANCEL "Event Notice" CALLED "Variable"	CANCEL TRANS CALLED TR

This statement removes a coming Endogenous Event from the calendar of coming Events. The Event may be rescheduled to occur at another time by another CAUSE statement, or the Event Notice may be destroyed or disposed of in any other desired manner. As in the case of the CAUSE statement, the "Variable" must be equal to the identification number of the particular Event Notice. The short form is also permissible.

GENERAL FORM	EXAMPLES
CANCEL "Event Notice" This short form is equivalent to: CANCEL "Event Notice" CALLED "Local Variable with same name as the Event Notice"	CANCEL ARRVL which means: CANCEL ARRVL CALLED ARRVL

FILE STATEMENTS

GENERAL FORM	EXAMPLES
FILE "Variable" IN "Set"	FILE ORDER IN QUE(MG) FILE ITEM IN LIST FILE JOB(MG) IN LOAD(MG,LC)

The FILE statement inserts an Entity into its proper position in a Set. The members of a Set are inserted and removed on a last–in–first–out (LIFO), first–in–first–out (FIFO), or ranked basis, depending on the type of Set organization specified on the Definition Form. The "Variable" in the FILE statement must be equal to the identification number of the Entity to be filed. The FILE statement automatically modifies the required member and owner Attributes.

The "Set" may have zero, one, or two subscripts, depending on whether it belongs to the System, a single Entity, or a pair of Entities. The subscripts are the identification numbers of the owning Entities. Ownership of double–subscripted Sets is limited to Permanent Entities.

REMOVE FIRST STATEMENTS

GENERAL FORM	EXAMPLES
REMOVE FIRST "Variable" FROM "Set"	REMOVE FIRST PART FROM BIN REMOVE FIRST X(M) FROM AB(J)

The REMOVE FIRST statement removes the first Entity from the Set, and the "Variable" is set equal to the identification of the particular Entity being removed. Determination of which Entity is first depends on the type of Set organization specified on the Definition Form. Modification of the required member and owner Attributes is automatically accomplished by the REMOVE FIRST statement.

REMOVE "SPECIFIC" STATEMENTS

GENERAL FORM	EXAMPLES
REMOVE "Variable" FROM "Set"	REMOVE TRUCK FROM SHOP REMOVE B(LM) FROM LSS

The REMOVE "SPECIFIC" statement removes a specified Entity from a Ranked Set irrespective of its rank position. The "Variable" must be equal to the identification number of the particular Entity to be removed. The identification number of the Entity must be in the Set at the time the REMOVE "SPECIFIC" statement is executed. Modification of the required owner and member Attributes is automatically accomplished by the REMOVE "SPECIFIC" statement.

The REMOVE "SPECIFIC" statement may not be used with LIFO or FIFO Sets. However, the same effect can be obtained by using a Ranked Set with the priority based on the time of entry into the Set. Using a Ranked Set to accomplish LIFO or FIFO operations should be avoided when possible, since Ranked Sets require more storage space and more computing time than do Sets with LIFO or FIFO organizations.

Chapter 5

ARITHMETIC AND CONTROL COMMANDS

SIMSCRIPT provides the following statements and phrases for performing and controlling arithmetic operations:

LET
STORE
FOR
FOR EACH "ENTITY"
FOR EACH OF "SET"
WITH

OR
AND
DO TO
LOOP
DO TO "SET"
REPEAT

LET STATEMENTS

GENERAL FORM	EXAMPLES
LET "Variable" = "expression","any number of control phrases separated by commas"	LET ALPHA = M(I) + COST LET SB(I) = C(I+E(I)), FOR EACH BASE I, WITH (H(X(I)))GR(5),AND (XYZ(I))LS(6)

The LET statement evaluates the "expression" and sets the "Variable" equal to this value. The expression may be integer or floating point, and may contain both Local and System Variables. If the mode of the expression is different from that of the Variable, the mode of the expression's value is automatically changed prior to setting the Variable equal to it.

The "Variable" to the left of the equal sign may or may not be subscripted. The subscript may be any integer expression and may in turn contain subscripted or non-subscripted integer Variables to any order.

In all SIMSCRIPT statements, written words or names must be separated by one or more spaces unless separated by a special character. Therefore, at least one blank space must be left between the word "LET" and the name of the "Variable." Any number of control phrases (FOR, FOR EACH "ENTITY," FOR EACH OF "SET," WITH, OR, or AND) may follow the expression to control the execution of the LET statement. These control phrases are described below.

STORE STATEMENTS

GENERAL FORM	EXAMPLES
STORE "Variable" IN "Variable","any number of control phrases separated by commas" At least one "Variable" must be a System Variable.	STORE ORDER IN ORDRA(ARRVL) STORE X(I) IN Y(AB(F))

The STORE statement sets the value of one Variable equal to the value of another Variable without regard to mode. In other words, whatever the binary representation of the value of the first Variable, the same binary representation is given to the value of the second Variable. This somewhat curious feature is provided so that Local Variables with either integer or floating point names may be used for storing Entity identification numbers.

The use of the STORE statement instead of the LET statement is mandatory when transferring an identification number from an integer Variable to a floating point Variable, or vice versa. To avoid errors, it is advisable always to use the STORE statement when transferring identification numbers.

CONTROL PHRASES

A number of SIMSCRIPT source language statements may be modified by what are called control phrases. These control phrases are:

FOR
FOR EACH "ENTITY"
FOR EACH OF "SET"
WITH
OR
AND

The statements that may be controlled by control phrases are:

LET
STORE
DO TO
DO TO "SET"
FIND MAX or MIN
FIND FIRST
CALL
READ
READ FROM
WRITE
COMPUTE

The general form of each kind of control phrase is described below, and its use with the various kinds of statements is described in the discussion of the individual statements. Control phrases are also used in the Report Generator discussed in Chapter 10.

FOR CONTROL PHRASES

GENERAL FORM	EXAMPLES
FOR "integer Local Variable" = ("integer expression")("integer expression")("integer expression") The last integer expression may be omitted.	FOR I = (1)(N) FOR I = (K)(N(X(I)))(2) FOR J = (2)(NBASE-1)

The first expression specifies the first value to which the Local Variable or index will be set. The second expression specifies the last or maximum value the Local Variable will assume. The third expression specifies the size steps by which the Local Variable is to be advanced. If this last expression is omitted, it is assumed to be "one." Each expression must be enclosed in parentheses.

In a sequence of FOR, FOR EACH "ENTITY," or FOR EACH OF "SET" phrases, the index of the last control phrase advances most rapidly and the index of the first control phrase advances most slowly.

FOR EACH "ENTITY" CONTROL PHRASES

Repetitive operations on each Permanent Entity of a particular type may be controlled by the FOR EACH "ENTITY" phrase.

GENERAL FORM	EXAMPLES
FOR EACH FOR ALL FOR EVERY	FOR EACH BASE I
"Permanent Entity" "integer Local Variable"	
which is equivalent to:	equivalent to:
FOR "integer Local Variable" = (1)(number of Permanent Entities of the particular type"	FOR I = (1)(NBASE) where NBASE equals the number of BASEs

EACH, ALL, and EVERY are synonyms. The translator automatically defines an unsubscripted Permanent Attribute with the name "NEntity type" for each type of Permanent Entity defined on the Definition Form. This System Attribute must be set equal to the number of Entities of the particular type each time the program is executed. For example, if BASE is a type of Permanent Entity, NBASE is automatically defined; it must be set equal to the number of BASEs each time the program is executed. The System Variable "NBASE" is available to all subprograms, but it may not be modified.

FOR EACH OF "SET" CONTROL PHRASES

Repetitive operations on each member of a Set may be controlled by the FOR EACH OF "SET" phrase.

GENERAL FORM	EXAMPLES
$\left.\begin{array}{l}\text{FOR EACH}\\ \text{FOR ALL}\\ \text{FOR EVERY}\end{array}\right]$ "Local Variable" $\left[\begin{array}{c}\text{OF}\\ \text{IN}\\ \text{ON}\\ \text{AT}\end{array}\right]$ "Set"	FOR EACH MEMBER IN QUE(J) FOR ALL I OF LIST FOR EVERY X OF S(I,M(J+K))

EACH, ALL, and EVERY are synonyms, and OF, IN, ON, and AT are synonyms. For ease in assigning names, the "Local Variable" may have either an integer or floating point name as long as the Variable is not involved in arithmetic operations in the statements controlled by the FOR EACH OF "SET" phrase. If the Local Variable is to be used in arithmetic operations, it must be given an integer name. As shown in the examples, the "Set" may or may not be subscripted.

WITH CONTROL PHRASES

GENERAL FORM	EXAMPLES
WITH ("expression")"comparison"("expression")	WITH (X+Z)GE(HCB) WITH (STOCK(I))LS(RP(I))

WITH phrases are used to limit FOR, FOR EACH "ENTITY" and FOR EACH OF "SET" phrases. Cases not satisfying the comparison indicated in the WITH phrase are excluded from the cases specified in the FOR, FOR EACH "ENTITY, " or FOR EACH OF "SET" phrase. Permissible comparisons and their codes are as follows:

Code	Comparison
GR	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LS	less than
LE	less than or equal to

The WITH phrase always limits the FOR, FOR EACH "ENTITY, " or FOR EACH OF "SET" phrase immediately preceding it. The two expressions in a WITH phrase need not be of the same mode; one may be integer and the other floating point.

OR CONTROL PHRASES

OR phrases impose alternative conditions on WITH phrases.

GENERAL FORM	EXAMPLES
`OR ("expression")"comparison"("expression)`	`OR (LEVEL + 1)LE(R + Q)`

Permissible "comparisons" and their codes are the same as described for the WITH phrase. The expressions may be integer or floating point and need not be of the same mode. Any number of OR phrases may modify a WITH phrase. OR phrases always modify the first WITH phrase immediately preceding them.

AND CONTROL PHRASES

AND phrases impose additional conditions on the previous WITH or OR phrase. Any number of AND phrases may modify WITH or OR phrases.

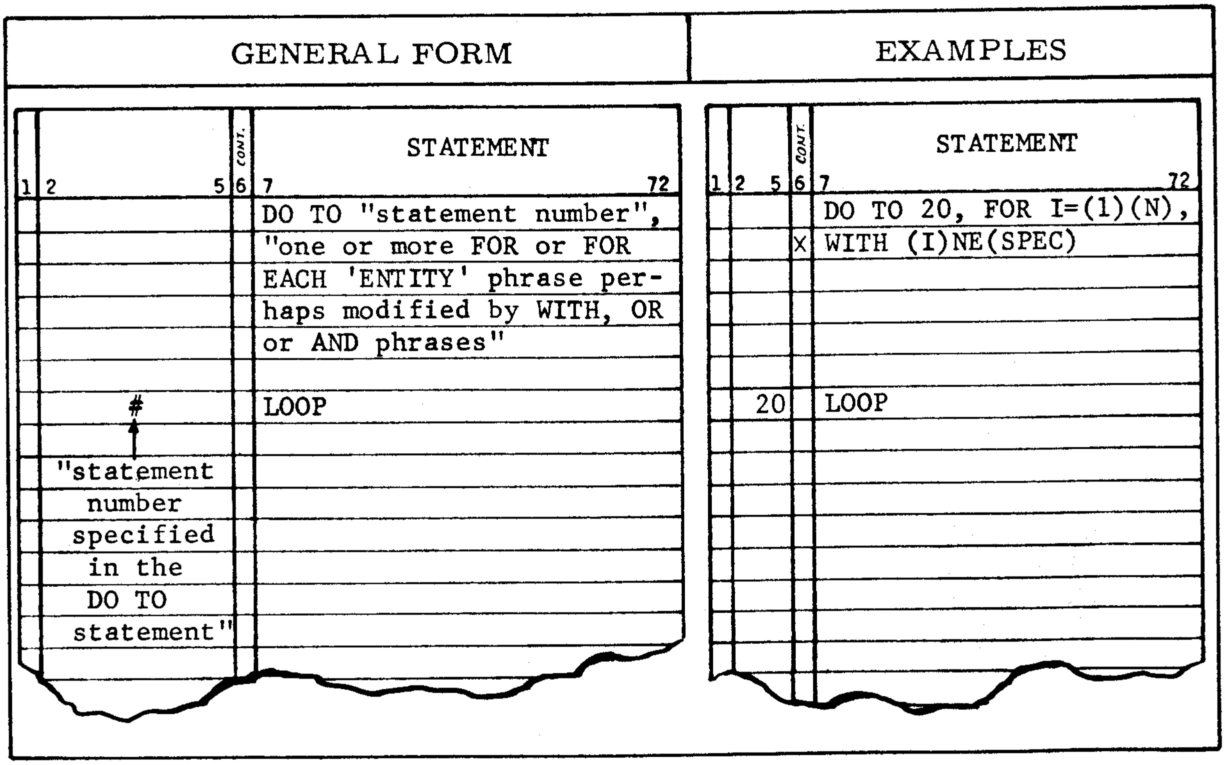
GENERAL FORM	EXAMPLES
`AND ("expression")"comparison"("expression")`	`AND (X(I))NE(Q(J))`

To summarize the use of control phrases, suppose it is desired to compute the total inventory value for a selected group of items in a supply system consisting of several different bases. Items are to be selected from a particular list, and only those items on the list which either have a unit price greater than $ 5000 or which have both a unit price greater than $1000 and an annual dollar volume of more than $ 5000 are to be included in the inventory value total. This computation could be accomplished by the following pair of statements:

```
        LET VALUE = 0.0

        LET VALUE = VALUE + PRICE(ITEM)*QUANT(ITEM, J),
                    FOR EACH BASE J, FOR EACH ITEM OF
                    LIST, WITH (PRICE(ITEM))GR(5000),
                    OR (PRICE(ITEM))GR(1000), AND
                    (PRICE(ITEM)*RATE(ITEM))GR(5000)
```

DO TO STATEMENTS AND LOOP STATEMENTS

The DO TO statement, and the LOOP statement always used in conjunction with it, are used to accomplish the repetitive execution of a series of statements. The DO TO statement is always controlled by one or more FOR or FOR EACH "ENTITY" phrases, perhaps modified by WITH, OR, or AND phrases. It may not be controlled by FOR EACH OF "SET" statements.



GENERAL FORM

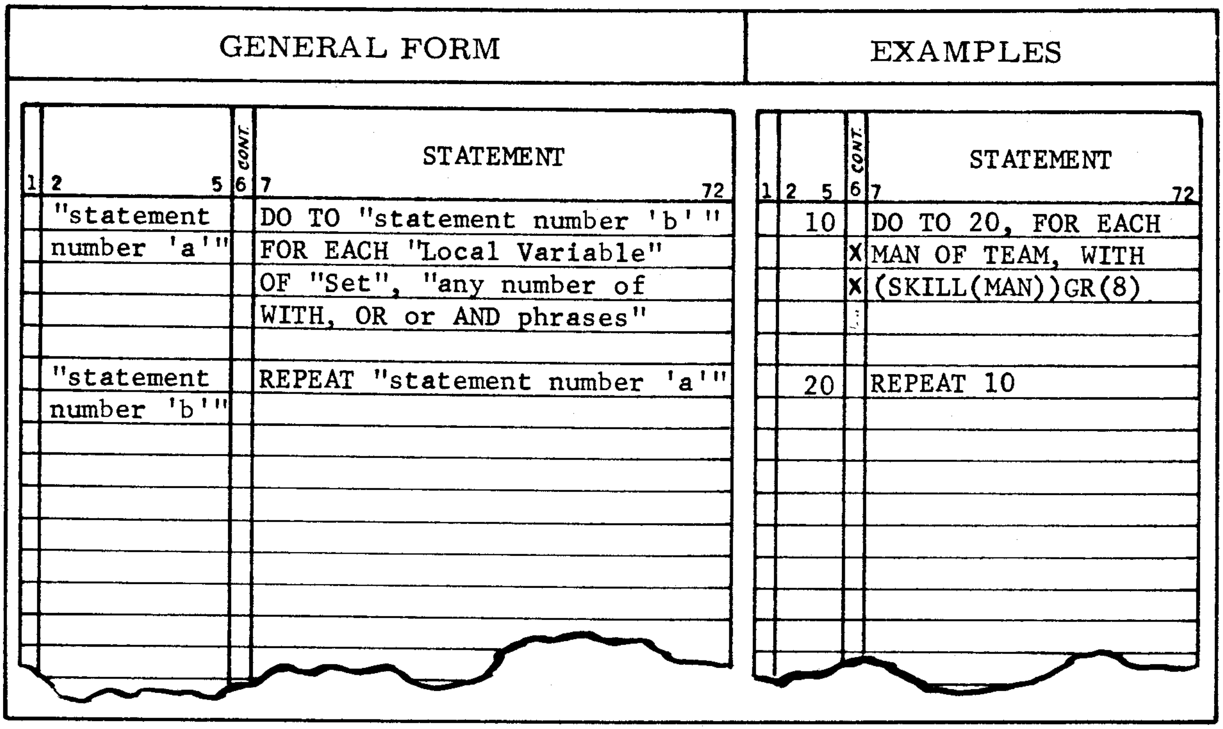
1	2 5	CONT. 6	STATEMENT 7 72
			DO TO "statement number",
			"one or more FOR or FOR
			EACH 'ENTITY' phrase per-
			haps modified by WITH, OR
			or AND phrases"
	#		LOOP
	"statement		
	number		
	specified		
	in the		
	DO TO		
	statement"		

EXAMPLES

1	2 5	CONT. 6	STATEMENT 7 72
			DO TO 20, FOR I=(1)(N),
		X	WITH (I)NE(SPEC)
	20		LOOP

DO TO "SET" STATEMENTS AND REPEAT STATEMENTS

If a series of statements are to be executed for each Entity in a Set, the DO TO "SET" statement must be used. The general forms of the DO TO "SET" statement and the REPEAT statement always used in conjunction with it are as follows:



GENERAL FORM

1	2 5	CONT. 6	STATEMENT 7 72
	"statement		DO TO "statement number 'b' "
	number 'a'"		FOR EACH "Local Variable"
			OF "Set", "any number of
			WITH, OR or AND phrases"
	"statement		REPEAT "statement number 'a'"
	number 'b'"		

EXAMPLES

1	2 5	CONT. 6	STATEMENT 7 72
	10		DO TO 20, FOR EACH
		X	MAN OF TEAM, WITH
		X	(SKILL(MAN))GR(8)
	20		REPEAT 10

In a DO TO "SET" statement, the last statement in its range must be a REPEAT statement returning control to the DO TO "SET" statement (it may never be a LOOP statement). Explicit statement numbers must be assigned to both the DO TO "SET" and REPEAT statements.

Only one FOR EACH OF "SET" phrase may be used to control each DO TO "SET" statement, but it may be modified by any number of WITH, OR or AND phrases.

Chapter 6

DECISION COMMANDS

SIMSCRIPT provides the following commands for selecting among alternatives:

IF
"THREE-WAY" IF
IF EMPTY
"UNCONDITIONAL" GO TO

"COMPUTED" GO TO
FIND MAX, FIND MIN
FIND FIRST
WHERE

IF STATEMENTS

GENERAL FORM	EXAMPLES

IF ("expression")"comparison"("expression"), "any statement"	IF (X+1)NE(Y), GO TO 50
	IF (STOCK(ITEM))EQ(0), LET NUMBER = NUMBER + 1

The IF statement makes the indicated comparison, and, if the condition is met, executes the indicated statement. If the condition is not met, the statement is ignored and control passes to the next statement in the program. Expressions may be of different mode. Permissible comparisons and their codes are:

Code	Comparison
GR	greater than
GE	greater than or equal to
EQ	equal to
NE	not equal to
LS	less than
LE	less than or equal to

"THREE-WAY" IF STATEMENTS

This SIMSCRIPT statement is like the IF statement in FORTRAN. If the expression is negative, control is transferred to the statement indicated by the first statement number. If the expression equals zero, control is transferred to the second statement. If the expression is positive, control is transferred to the third statement. As in other SIMSCRIPT

GENERAL FORM	EXAMPLES
IF ("expression") "statement number", "statement number", "statement number"	IF (PRITY(K)+B) 10, 20, 30

statements, the expressions may contain both Local and System Variables and may have subscripted subscripts.

IF EMPTY STATEMENTS

GENERAL FORM	EXAMPLES

IF "Set" $\left\{\begin{matrix}\text{IS}\\ \text{IS NOT}\end{matrix}\right\}$ EMPTY, "any statement"	IF QUE(MG) IS EMPTY, GO TO 50
	IF LIST(A) IS NOT EMPTY, REMOVE FIRST ITEM FROM LIST(A)

The preceding statement is included in the SIMSCRIPT language as a convenient way to test whether a Set is empty. If the condition is met, the indicated statement is executed. If the condition is not met, the statement is ignored and control passes to the next statement in the program.

"UNCONDITIONAL" GO TO STATEMENTS

GENERAL FORM	EXAMPLES
GO TO "statement number"	GO TO 40

The execution of this statement transfers program control to the statement with the specified statement number.

"COMPUTED" GO TO STATEMENTS

The execution of this statement transfers program control to either the first, second, third, etc., statement depending on whether the expression

GENERAL FORM	EXAMPLES
GO TO ("statement number", "statement number", ..."statement number"), "expression"	GO TO (30, 20, 95), I + J

equals 1, or 2, or 3, etc. This statement is the same as in FORTRAN except that where FORTRAN requires an integer variable, SIMSCRIPT permits any expression. Floating point expressions will be truncated.

FIND MAX, FIND MIN STATEMENTS

GENERAL FORM	EXAMPLES
FIND "Variable" = $\left\{\begin{matrix}\text{MAX}\\ \text{MIN}\end{matrix}\right\}$ OF "expression", "one or more control phrases, and if desired, WHERE phrases", IF NONE, "any statement"	FIND MSTOCK = MAX OF STOCK(I), FOR EACH ITEM I FIND MPRI = MIN OF PRI(MG)* FCTR(LC), FOR EACH LC OF LIST2, WHERE LCMIN IS BEST LC, FOR EACH MG OF SRUD(LC), WITH (NIDL(MG))GR(0), WHERE MGMIN IS BEST MG, IF NONE, GO TO 50

This statement finds the maximum or minimum value of an expression and sets a Variable equal to this value. The search is controlled by one or more FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrases, perhaps modified by WITH, OR, or AND phrases. Additionally, any FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" may have one WHERE phrase associated with it. The WHERE phrase is described below.

Cases are possible in which there is no maximum or minimum—for example, when a FOR EACH OF "SET" phrase refers to a Set that happens to be empty. It can also occur that the conditions of WITH, OR, or AND phrases cannot be fulfilled. The "IF NONE" phrase permits the specification of a SIMSCRIPT statement to be executed if the MAX or MIN is found to be nonexistent. The "IF NONE" phrase may be omitted if there is no possibility that the MAX or MIN does not exist.

WHERE PHRASES

GENERAL FORM	EXAMPLES

WHERE "Variable" "optional text"	WHERE MGMIN IS BEST MACHINE GROUP
	or optionally:
	WHERE MGMIN

Being a selection statement, FIND MAX or MIN produces at least two values: (1) the value of the maximum or minimum result, and (2) the identification of the particular Variable or Variables that produced this result. The "Variable" in the WHERE phrase will be set equal to the value attained by the index or identification number controlled by the preceding FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrases when the minimum or maximum is achieved. The WHERE phrase refers to the preceding FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrase, even though it may be separated from it by WITH, OR, or AND phrases. This is illustrated in the following example:

```
FIND XY = MAX OF X(I)*Y(I), FOR EACH I OF LIST,
          WITH (X(I))GR(0), WHERE IMAX IS THE
          VALUE OF I WHICH MAXIMIZES X(I)*Y(I)
```

As stated above, the text following "WHERE IMAX" is optional. There are no restrictions on the optional text except that it may not contain commas.

FIND FIRST STATEMENTS

GENERAL FORM	EXAMPLES
FIND FIRST "optional text", "one or more control phrases and WHERE phrases", IF NONE, "any statement"	FIND FIRST, FOR EACH SN I, WITH (X(I))GR(0), WHERE IFIRST IS FIRST, IF NONE, GO TO 60 FIND FIRST COMBINATION OF I AND J, FOR I = (1)(N), WHERE IX, FOR J = (1)(N2), WHERE JX, WITH (X(I))LE(5), AND (Y(J))GR(7)

In the first example, "IFIRST" is set equal to the first value of "I" with X(I) greater than zero. The second example finds the first combination of "I" and "J" that meets the specified conditions. It sets "IX" equal to "I", and "JX" equal to "J" for this combination. It is not necessary that each FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrase have a WHERE phrase, and the IF NONE phrase may be omitted. However, there must be either a WHERE phrase or an IF NONE phrase in the FIND FIRST statement.

If there are several FOR, FOR EACH "ENTITY", and FOR EACH OF "SET" phrases, the FIND FIRST statement selects the first combination of Variables encountered in the search which meets all the specified conditions. The Variables indicated in the WHERE phrases are then set equal to the values of the indexes in their corresponding FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrases.

Chapter 7

INPUT AND OUTPUT COMMANDS

In SIMSCRIPT programs, the input of data is usually done by means of the initialization procedures and subsequently from the Exogenous Events Tape. Printed output is usually obtained by means of the Report Generator. A number of tape–handling commands are also provided.

The present Chapter describes the composition of the Exogenous Events Tape plus the following tape–handling commands:

SAVE
READ
FORMAT
READ FROM
WRITE ON
ADVANCE
BACKSPACE
REWIND
ENDFILE
LOAD
RECORD MEMORY
RESTORE STATUS

Initialization procedures and the Report Generator are described in subsequent Chapters.

EXOGENOUS EVENT TAPE

The Exogenous Event Tape is formed from a deck of cards containing one Exogenous Event Card for each Event occurrence. Exogenous Events are ordered chronologically on the tape, and each Event card may be followed by one or more Data Cards for input of information related to the Event. Exogenous Event Cards have the following layout:

Cols. 1–3	identification number of the Exogenous Event type
Cols. 4–7	the day of simulation that the Event occurs
Cols. 8–10	the hour of the day that the Event occurs
Cols. 11–12	the minute of the hour that the Event occurs
Cols. 13–72	additional data, if any.

The data in Cols. 1 through 12 of the Event Card are available only to the SIMSCRIPT–generated Timing routine and may not be referred to in source language statements. On the basis of these data, the Timing routine advances simulated time and calls the appropriate Exogenous Event routine.

Data to be read in by the Exogenous Event routine may be entered in Cols. 13 through 72 of the Event Card, or may be entered in Cols. 1 through 72 of subsequent Data Cards.

SAVE STATEMENTS

If data are to be read from Cols. 13 through 72 of the Event Card, the second statement in the Exogenous Event routine must be a SAVE statement. If the second statement is not a SAVE statement, these columns are ignored, and the next-executed READ statement will start reading from the first column of the following Data Card. The SAVE statement is used only for this one purpose; its general form is as follows:

GENERAL FORM	EXAMPLES
SAVE "optional text"	SAVE SAVE EVENT CARD

READ STATEMENTS

Data are read from the Exogenous Event Tape by means of a pair of READ and FORMAT statements.

GENERAL FORM	EXAMPLES
READ "any number of Variables separated by commas", "any number of FOR or FOR EACH 'ENTITY' phrases"	READ RL, RB, M READ X(I), Y(I), FOR I=(1)(N)

The Variables in the READ statement may be Local or System, subscripted or unsubscripted. If a READ statement contains both subscripted and unsubscripted Variables controlled by FOR or FOR EACH "ENTITY" phrases, the unsubscripted Variables will be read in repeatedly until the control phrases are satisfied. In addition to FOR and FOR EACH "ENTITY" control phrases, FOR EACH OF "SET", WITH, OR, and AND are also permitted in controlling a READ statement. However, as a practical matter they would rarely be used since data entries on the tape must correspond precisely to the conditions called for in the control phrases.

If data are being read from an Event Card, the READ statement begins reading from Col. 13; otherwise it begins reading from Col. 1.

The READ statement may be used only to read from the Exogenous Event Tape. Data are read from other tapes by means of the READ FROM statement.

FORMAT STATEMENTS

A FORMAT statement must immediately follow each READ, READ FROM, or WRITE statement.

GENERAL FORM	EXAMPLES
FORMAT "optional constant" ("one field description for each Variable in the preceding statement")	FORMAT (I6,2D3.5) FORMAT 6(2I6)

Field descriptions are separated by commas and correspond in order to the Variables mentioned in the preceding READ, READ FROM, or WRITE statement.

The "optional constant" may be used to indicate that the group of Variables specified in the preceding READ, READ FROM, or WRITE statement is repeated a certain number of times per card or record. For example,

```
READ X(I), Y(I), FOR I = (1) (N)
FORMAT 6(2I5)
```

indicates that up to six pairs of X(I) and Y(I) are on each card, and that each Variable is contained in a five–digit integer field. If the "optional constant" is omitted, it is assumed to be "one."

When reading from the Exogenous Event Tape, no more than 72 characters may be specified in a FORMAT statement (no more than 60 if data are being read from an Event Card). In other words, the number of characters called for in the field descriptions times the optional constant, if any, must not exceed 72 (or 60 in the case of an Event Card). The number of characters specified in FORMAT statements accompanying READ FROM or WRITE statements has no limit except that it may not exceed the length of the tape record. Six different types of field descriptions may be used in FORMAT statements:

Integer
Decimal
Decimal Hours
Days, Hours, and Minutes
Alpha–Numeric
Skip

Integer Field Description

GENERAL FORM	EXAMPLES
aIb where a = number of consecutive fields b = number of digits	6I5 2I4 I8

If there is to be only one field of a particular configuration, the "1" may be omitted as in the third example above. The number of digits must always be indicated even if the field consists of a single digit. An integer field may be of any size but only the last six digits will be read in. The value is stored in the decrement in the standard FORTRAN manner.

Decimal Field Description

GENERAL FORM	EXAMPLES
aDb.c where a = number of fields b = number of columns in front of the decimal point c = number of columns following the decimal point	2D6.3 4D4.4 D3.6

If the "number of fields" designation is omitted, it is assumed to be "one."

When data are punched into a decimal field for transfer to a tape, the decimal point must always be punched. Therefore, the width of the field will be one character greater than the total number of characters specified to precede and follow the decimal point. For example, the field description "D3.6" would use up 10 columns on an Event Card and 10 characters on the Event Tape. A decimal number may not have more than six digits on either side of the decimal point.

Decimal–Hours Field Description

Within a SIMSCRIPT program, simulated time is always expressed in days and fractions of days. However, it is sometimes convenient to input time–values expressed in hours and fractions of hours, or as days, hours, and minutes. Consequently, special field descriptions are provided for this purpose. (Decimal–days are read in using the previously described decimal field description.)

The decimal–hours field description has the following form:

GENERAL FORM	EXAMPLES
aHb.c where a = number of fields b = number of columns in front of the decimal point c = number of columns following the decimal point	4H3.2 H2.1

This field description follows the same rules as those described for the decimal field description except that values input as decimal–hours are automatically converted to decimal–days.

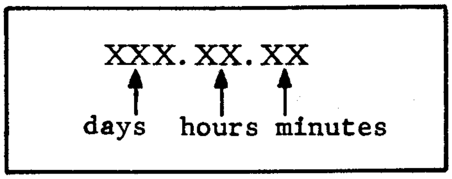
In converting from decimal–hours to decimal–days, the translator assumes 24 hours per simulated day. If a different number of hours per day is desired, the System Attribute called "HOURS" may be modified by source language statements, or it may be given a different value as part of the initial conditions. The System Attribute HOURS is automatically defined by the translator and equals the number of hours per day (see Chapter 11 for further discussion.)

Days, Hours, and Minutes Field Description

There are three possible variations to the "days, hours, and minutes" field description.

GENERAL FORM	EXAMPLES
aMb aMc.b aMd.c.b where a = number of fields b = number of minutes columns c = number of hours columns d = number of days columns	2M3 M6.2 4M3.2.2

As before, if the number of fields is omitted it is assumed to be "one." As shown above, it is possible to omit "days" or "days and hours" from the field description. For example, "M6. 2" specifies a single field, six columns allocated to "hours," and two columns allocated to "minutes." 2M3 means two fields, each with three columns allocated to minutes only. In punching, columns allocated to "days" must be separated by a period from columns allocated to "hours," and "hours" columns from "minutes" columns. For example, the field description "M3. 2. 2" would require nine characters as shown below:



Time values input as days, hours, and minutes are automatically converted to decimal days. In making the conversion, the object program uses the values of HOURS and MINS which are automatically defined System Attributes, and are equal to the number of hours per day and the number of minutes per hour respectively. The values of HOURS and of MINS may be specified in the initialization procedures and may be modified by source language statements. Unless otherwise specified, HOURS is automatically set equal to 24.0 and MINS to 60.0 at the start of object program execution. Time–values input in terms of decimal days may be expressed to six–decimal–place accuracy. Decimal–hours will be rounded to decimal–days with six–place accuracy. The accuracy limit for days, hours and minutes is, of course, one minute.

Alpha–Numeric Field Description

If it is desired to store alpha–numeric information so as to output it later, the following field description may be used:

GENERAL FORM	EXAMPLES
aAb where a = number of fields b = number of characters in the field (must be equal to or less than six)	2A6 A5

Since each alpha–numeric field must be stored in one full storage word, the number of fields also equals the number of storage words to be used. If an alpha–numeric field contains more than six characters, only the last six characters will be read in.

Skip Description

If it is desired to skip over any part of a tape record, the following instruction or field description may be used:

GENERAL FORM	EXAMPLES
Sb where b = number of columns or characters to be skipped	S12 S2

Right-adjusted Integer Field Description

GENERAL FORM	EXAMPLES
aJb where a = number of consecutive fields b = number of digits	2J6 J3

This field description reads an integer into the address portion of a word (in contrast with the standard FORTRAN integer which is stored in the decrement). The value may not exceed six digits although the field may consist of more than six columns. This field description may be used to read in alternate values for the random root used to generate random numbers as discussed in Chapter 11. These values must be stored in a full word ("packing" as discussed in Chapter 13 is not permitted).

READ FROM STATEMENTS

GENERAL FORM	EXAMPLES
READ FROM TAPE] READ FROM] "expression","any number of Variables", "any number of FOR or FOR EACH 'ENTITY' phrases"	READ FROM TAPE 5, A(I), B(I), C(I), FOR I = (1)(N) READ FROM N+1, CASE, RUN

The READ FROM statement reads data from any specified tape other than the Exogenous Event tape. The "expression" must equal the tape number.

The word "TAPE" may be omitted when writing the statement. A FORMAT statement must immediately follow each READ FROM statement.

In addition to FOR and FOR EACH "ENTITY" phrases, the other control phrases (FOR EACH OF "SET", WITH, OR, and AND) may also be used, but only if the data being read are certain to correspond to the conditions specified by the control phrases.

WRITE STATEMENTS

GENERAL FORM	EXAMPLES
WRITE ON TAPE WRITE ON] "expression", "any number of Variables", "any number of FOR or FOR EACH 'ENTITY' phrases"	WRITE ON 6, A, B, C WRITE ON TAPE K-1, QUE(I,J), FOR EACH MG I, FOR EACH PLANT J

The WRITE statement writes data onto any specified tape other than the Exogenous Event tape. The value of the "expression" indicates the tape number. The word "TAPE" may be omitted. A FORMAT statement must accompany each WRITE statement. The control phrases may be used if appropriate.

ADVANCE STATEMENTS

Tape units may be advanced by the following statement:

GENERAL FORM	EXAMPLES
ADVANCE TAPE ADVANCE] "expression","expression" { RECORD RECORDS FILE FILES	ADVANCE TAPE N-1,M RECORDS ADVANCE 3, 1 FILE

It is necessary to distinguish between Records and Files. The word "TAPE" may be omitted. The value of the first "expression" specifies the tape number. The value of the second "expression" specifies the number of records or the number of end–of–file marks through which the tape is to be advanced.

BACKSPACE STATEMENTS

GENERAL FORM	EXAMPLES
$\left.\begin{array}{l}\text{BACKSPACE TAPE}\\\text{BACKSPACE}\end{array}\right]$ "expression", "expression" $\left\{\begin{array}{l}\text{RECORD}\\\text{RECORDS}\\\text{FILE}\\\text{FILES}\end{array}\right.$	BACKSPACE TAPE 2, 4 RECORDS BACKSPACE J + I, N-1 FILES

The word "TAPE" is optional. The first "expression" indicates the tape number. The second "expression" indicates the number of records or end–of–file marks through which the tape is to be backspaced. In the case of files, the tape will be positioned at the beginning of the file adjacent to the last end–of–file mark specified in the BACKSPACE statement. For example, if a tape is positioned at the beginning of a file, "backspacing one file" will leave the tape where it is, since the tape will be backspaced through the adjacent end–of–file mark and then advanced to its original position at the beginning of the file.

REWIND STATEMENTS

A tape may be rewound by the following statement:

GENERAL FORM	EXAMPLES
$\left.\begin{array}{l}\text{REWIND TAPE}\\\text{REWIND}\end{array}\right]$ "expression"	REWIND 11 REWIND TAPE N+1

The "expression" specifies the tape number. The word "TAPE" may be omitted.

ENDFILE STATEMENTS

An end—of—file mark may be written on a tape by the following statement:

GENERAL FORM	EXAMPLES

ENDFILE TAPE
ENDFILE
"expression"
	ENDFILE 12

The word "TAPE" may be omitted. The "expression" specifies the tape number.

LOAD STATEMENT

GENERAL FORM	EXAMPLES
LOAD PROGRAM LOAD "integer expression"	LOAD PROGRAM FROM TAPE 3

FROM TAPE
FROM
"integer expression"
	LOAD I+1 FROM 2

The LOAD statement loads a designated program from a designated tape. The first "expression" specifies the identification number of the particular program to be loaded. This identification number may have any value from 1 to 32,767. The second "expression" specifies the logical designation of the tape, and may take on only the values 2, 3, or 4. At the time of program execution, both the program identification number and the tape number must also be specified on a "CHAIN (R, T)" control card as required by the FORTRAN Monitor. (See Chapter 12 for a discussion of SIMSCRIPT Compile and Execute Decks.)

RECORD MEMORY STATEMENTS

Except for a few words not available to the programmer, the entire contents of memory followed by an end—of—file mark may be written on tape by means of the RECORD MEMORY statement. A particular picture of memory may thus be preserved and subsequently read back in by means of the RESTORE STATUS statement.

GENERAL FORM	EXAMPLES
RECORD MEMORY ON TAPE "expression", RESTORE TO "statement number"	RECORD MEMORY ON TAPE 9, RESTORE TO 30

Following execution of the RESTORE STATUS statement, control will go to the statement specified in the original RECORD MEMORY statement, and execution will continue, based on the newly restored contents of memory.

RESTORE STATUS STATEMENTS

GENERAL FORM	EXAMPLES
RESTORE STATUS FROM TAPE "expression"	RESTORE STATUS FROM TAPE 9

The RESTORE STATUS statement will replace the current contents of memory with the previously recorded picture of memory (except for a few words not available to the programmer). It also restores the contents of index registers 1 and 2 and all four sense lights. Execution will proceed with the statement specified at the time the contents of memory were originally recorded on tape. The tape file containing the previous picture of memory must be backspaced into position prior to execution of the RESTORE STATUS statement. The value of the "expression" specifies the number of the tape. The word "TAPE" may be omitted from the RESTORE STATUS and RECORD MEMORY statements.

Chapter 8

MISCELLANEOUS COMMANDS

Refusing to fit gracefully into the organization of previous Chapters, the following statements and topics are presented here:

ACCUMULATE
COMPUTE
STOP
DIMENSION
FORTRAN INSERTS
STATEMENT CONTINUATION

ACCUMULATE STATEMENTS

In simulation problems, it is usually necessary to accumulate various totals as a function of time. Since status in SIMSCRIPT can change only at points in time called Events and the time between Events may be variable, a typical cumulative total might be represented by the area under the curve shown in Fig. 12.

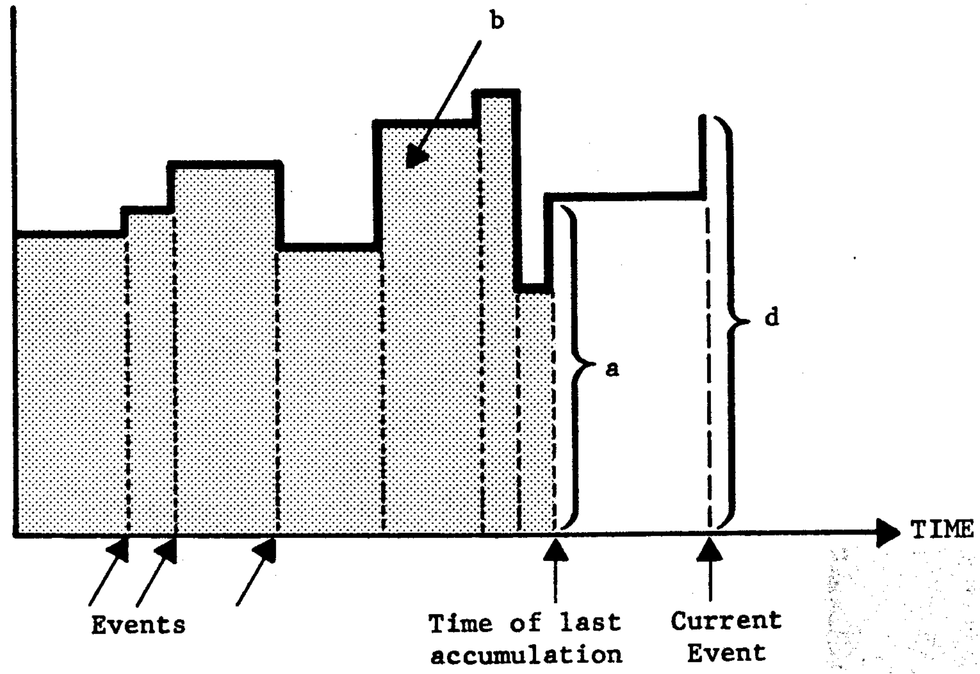


Fig. 12 — Illustration of Cumulative Total

Any number of cumulative totals of this type can be obtained by the ACCUMULATE statement shown below.

GENERAL FORM	EXAMPLES
$\left.\begin{matrix}\text{ACCUMULATE}\\ \text{ACC}\end{matrix}\right]$ "floating point Variables $a_1, a_2, \ldots a_n$" INTO "floating point Variables $b_1, b_2, \ldots b_n$" SINCE "floating point Variables $c_1, c_2, \ldots c_n$", "any number of POST and/or ADD phrases separated by commas The POST and ADD phrases have the following form: $\left.\begin{matrix}\text{POST}\\ \text{ADD}\end{matrix}\right\}$"floating point expressions $d_1, d_2, \ldots d_m$"	ACCUMULATE IDLE(MG), WAIT(ORDER) INTO CIDLE(MG), CWAIT(ORDER) SINCE LASTM(MG), LASTO(ORDER), ADD -1.0, POST NEWAT

The ACCUMULATE statement multiplies the Variables a_i by the elapsed times since the time values c_i, and adds these quantities to the Variables b_i. It then updates the time Variables c_i. Finally, it either adds each expression d_i to the Variable a_i or sets the Variables a_i equal to the expressions d_i.

More specifically, for each group of Variables a_i, b_i, c_i, and expression d_i, the ACCUMULATE statement accomplishes the following computations in this order:

$$(1) \qquad b_i = b_i + a_i \cdot (\text{current TIME} - c_i)$$

$$(2) \qquad c_i = \text{current TIME}$$

$$(3) \qquad \begin{cases} a_i = d_i, \text{ if POST} \\ a_i = a_i + d_i, \text{ if ADD} \end{cases}$$

The Variables and expressions must all be floating point. The Variables may be Local or System. The word "ACCUMULATE" may be abbreviated to "ACC."

The expressions d_i in POST and/or ADD phrases correspond in order to the Variables a_i. If the number (k) of expressions d_i appearing in all POST and ADD phrases is less than the number (n) of Variables a_i, the last (n–k) Variables are not modified by the ACCUMULATE statement. The ACCUMULATE statement may also be written without POST or ADD phrases. If all "time" Variables c_1, c_2, ... c_n always have the same value, the following variation of the ACCUMULATE statement may be used.

GENERAL FORM	EXAMPLES
ACCUMULATE ACC] "Variables $a_1, a_2, \ldots a_n$" INTO "Variables $b_1, b_2, \ldots b_n$" ALL SINCE "Variable", "POST or ADD phrases"	ACC B(X), C(X), D(Y), INTO R(X), S(X), T(Y), ALL SINCE TM, ADD 1.0

COMPUTE STATEMENTS

The following command is provided to permit the easy computation of certain frequently needed statistical quantities.

GENERAL FORM	EXAMPLES
COMPUTE "one to seven Variables" = "one to seven statistical quantities" OF "expression", "any number of control phrases"	COMPUTE MX, SX, VX = MEAN, STD-DEV, VARIANCE OF X(I), FOR EACH BASE I

Variables on the left side of the equation may be Local or System Variables. For each Variable indicated on the left side, a statistical quantity must be indicated on the right side. Variables and statistical quantities correspond in order, one for one. The indicated statistical quantities are computed for the indicated expression across the particular subscripts specified by the control phrases.

The source language names and definitions of the permissible statistical quantities are as follows:

Name		Definition
NUMBER	=	number of cases that meet the conditions specified by the control phrases
SUM	=	Σ expression
MEAN	=	$\dfrac{\Sigma \text{ expression}}{\text{NUMBER}}$

$$\text{SUM-SQUARES} = \Sigma\,(\text{expression})^2$$

$$\text{MEAN-SQUARE} = \frac{\Sigma(\text{expression})^2}{\text{NUMBER}}$$

$$\text{VARIANCE} = \frac{\Sigma(\text{expression} - \text{MEAN})^2}{\text{NUMBER}} = (\text{MEAN-SQR})-(\text{MEAN})^2$$

$$\text{STD-DEV} = \sqrt{\text{VARIANCE}}$$

STOP STATEMENTS

GENERAL FORM	EXAMPLES
STOP	STOP

This statement stops the execution of the program. It is equivalent to the FORTRAN statement CALL EXIT.

DIMENSION STATEMENTS

GENERAL FORM	EXAMPLES
DIMENSION "any number of subscripted Local Variables with each subscript equal to its maximum value"	DIMENSION A(5), B(20), X(5,50)

If Local Variables are to be subscripted they must appear in a DIMENSION statement. The "subscripts" in the DIMENSION statement are integer constants, indicating the maximum value the subscript can take on.

FORTRAN INSERTS

In addition to the previously described source language, the FORTRAN language is also available for operations with Local Variables except that COMMON, FREQUENCY, and IF OVERFLOW statements are not permitted. Statements written in FORTRAN may not refer to SIMSCRIPT System Variables. FORTRAN statements may be interspersed among SIMSCRIPT statements in any desired manner. The first column of the coding sheet is used to distinguish between SIMSCRIPT statements and FORTRAN statements according to the following rules. If Col. 1 is blank, it signifies a SIMSCRIPT statement; if it contains an "X," it signifies an ordinary FORTRAN statement and will be translated as if the FORTRAN statement contained a blank in Col. 1. All other kinds of FORTRAN statements requiring a code letter in Col. 1 may be written exactly as in FORTRAN (namely, C for Comment, B for Boolean, I for Imaginary, F for Function, D for Double Precision, and S for SAP). The first column on the coding form may not be used for any other purpose, and statement numbers are therefore limited to four digits.

A number of statements such as GO TO, SUBROUTINE, "THREE-WAY" IF, etc. are the same in SIMSCRIPT as they are in FORTRAN. Such statements, written with a blank in Col. 1, have access to System Variables and may use subscripted subscripts within expressions.

STATEMENT CONTINUATION

Statements may be continued from one line to the next of the coding form by inserting a mark in Col. 6. A statement may be continued on as many as 99 successive lines.

Control phrases such as FOR, WITH, etc. which may follow statements such as LET, FIND, etc. are considered part of the statement being controlled. If control phrases appear on a subsequent line, a mark is required in Col. 6.

The statement following an IF statement or an IF NONE control phrase in a FIND statement is considered to be part of the IF or FIND statement, and requires a continuation mark if it appears on a subsequent card.

Chapter 9

EVENT ROUTINES AND OTHER SUBPROGRAMS

SIMSCRIPT programs consist of a number of different kinds of both written and generated subprograms. The present chapter discusses:

Exogenous Event Routines
Endogenous Event Routines
Subroutines
Functions
Events List
System Package
Entity, Attribute, and Set Packages
Main Routine

In addition to these subprograms, output routines are generated by the SIMSCRIPT Report Generator as discussed in Chapter 10.

EXOGENOUS EVENT ROUTINES

Most of the special features of Exogenous Event routines have been discussed in previous Chapters. They are summarized here to provide a single point of reference.

Exogenous Event routines are called by the Timing routine in response to an entry on the Exogenous Event Tape. Prior to calling an Exogenous Event routine, the Timing routine automatically advances simulated time to the value specified for the particular Event occurrence. Exogenous Event routines may not be called by any other means.

The Exogenous Event Tape is formed from a deck of cards. There must be one Event Card for each occurrence of each kind of Exogenous Event. The Event Cards are presented in the order of their occurrence. Each Event Card contains an identification number indicating what kind of Exogenous Event it represents. It also contains the day, hour, and minute in simulated time that the Event is to occur.

Additional data describing the Event may be included in Cols. 13 to 72 of the Event Card. Any number of additional Data Cards may also follow the Event Card on the Event Tape. The Event Card format was described in Chapter 7.

The correspondence between the identification number and the name of each type of Exogenous Event is established by the Events List described later in the present Chapter.

The first statement of each Exogenous Event routine must be an EXOGENOUS EVENT statement:

GENERAL FORM	EXAMPLES
$\left.\begin{array}{l}\text{EXOGENOUS}\\ \text{EXOG}\end{array}\right]$ EVENT "Event Name"	EXOGENOUS EVENT SALE EXOG EVENT DEMAND

"EXOGENOUS" may be abbreviated to "EXOG" as shown above. The Event name may consist of any one to six letters or numbers, the first character must be a letter, and the name may not contain "XX" or end in "F."

If data are to be read from the Event Card, the second statement in the Exogenous Event routine must be a SAVE statement. The first READ statement after a SAVE statement will start reading in Col. 13 of the Event Card. If the SAVE statement is omitted, the next READ statement will start reading in Col. 1 of the Data Card immediately following the Event Card. The READ statement may be used only for reading from the Exogenous Event Tape. The READ FROM statement may be used to read from other tapes.

In Exogenous Event routines, control is returned to the Timing Routine by the RETURN statement shown below.

GENERAL FORM	EXAMPLES
RETURN	RETURN

An END statement must appear at the physical end of each Exogenous Event routine:

GENERAL FORM	EXAMPLES
END	END

ENDOGENOUS EVENT ROUTINES

Endogenous Event routines are called by the Timing routine as a result of CAUSE statements in previously executed Event routines or Subroutines. Prior to calling an Endogenous Event routine, the Timing routine advances simulated time to the value specified in the precipitating CAUSE statement. Endogenous Event routines may not be called in any other manner. The first statement in each Endogenous Event routine must be an ENDOGENOUS EVENT statement:

GENERAL FORM	EXAMPLES

ENDOGENOUS
ENDOG] EVENT "name of Event Notice"
	ENDOGENOUS EVENT EPROC
	ENDOG EVENT START

"ENDOGENOUS" may be abbreviated to "ENDOG" as shown.

The Event name must be identical to the name of a type of Event Notice defined on the Definition Form. This same name is also automatically given to a Local Variable in the Endogenous Event routine, and this Local Variable is automatically set equal to the identification number of the Event Notice as previously specified in the CAUSE statement. The name of each Endogenous Event must also appear in the Events List described later in this Chapter.

Control is returned to the Timing Routine by the RETURN statement, and an END statement must appear at the physical end of each Endogenous Event routine.

SUBROUTINE SUBPROGRAMS

The conventions for writing Subroutine subprograms in SIMSCRIPT are the same as in FORTRAN, except as noted below.

The SUBROUTINE statement must be the first statement in the subprogram:

GENERAL FORM	EXAMPLES
SUBROUTINE "Subroutine name" ("arguments, if any")	SUBROUTINE ALLOC(LC)

This statement must appear as the first statement of all subprograms with the following exceptions. It may not be used to begin subprograms describing Endogenous Events, Exogenous Events, Report routines, the Events List, Functions, or the "MAIN" routine in non–simulation programs. Arguments appearing in a SUBROUTINE statement must be Local Variables and are separated by commas.

Control is returned to the calling routine by the RETURN statement, and an END statement appears at the physical end of each Subroutine subprogram.

CALL STATEMENTS

Subroutines are called by means of the CALL statement:

GENERAL FORM	EXAMPLES
CALL "Subroutine name" ("arguments, if any") "any number of control phrases separated by commas"	CALL INTER(I+J,*X(I)) FOR I = (1)(N) CALL SHIP

If the value of the argument is to be given to the Subroutine by the calling routine, the argument may be an expression. If, on the other hand, the value of the argument is to be given by the Subroutine to the calling rou–

tine it may be any Variable and it must be preceded by an asterisk. If the argument is to be given to the Subroutine by the calling routine and then given back to the calling routine with perhaps a different value, it must be a Variable and must be preceded by two asterisks.

The CALL statement may be followed by any number of control phrases separated by commas.

FUNCTION SUBPROGRAMS

Except as noted below, Function subprograms are handled in the same manner in SIMSCRIPT as in FORTRAN. The name and mode of each Function subprogram is specified on the Definition Form, and Function names are formed according to the same rules as were described for System Variables.

The first statement in a Function subprogram is the FUNCTION statement:

GENERAL FORM	EXAMPLES
FUNCTION "Function name" ("arguments, if any")	FUNCTION DELAY FUNCTION NXQ(X,Y)

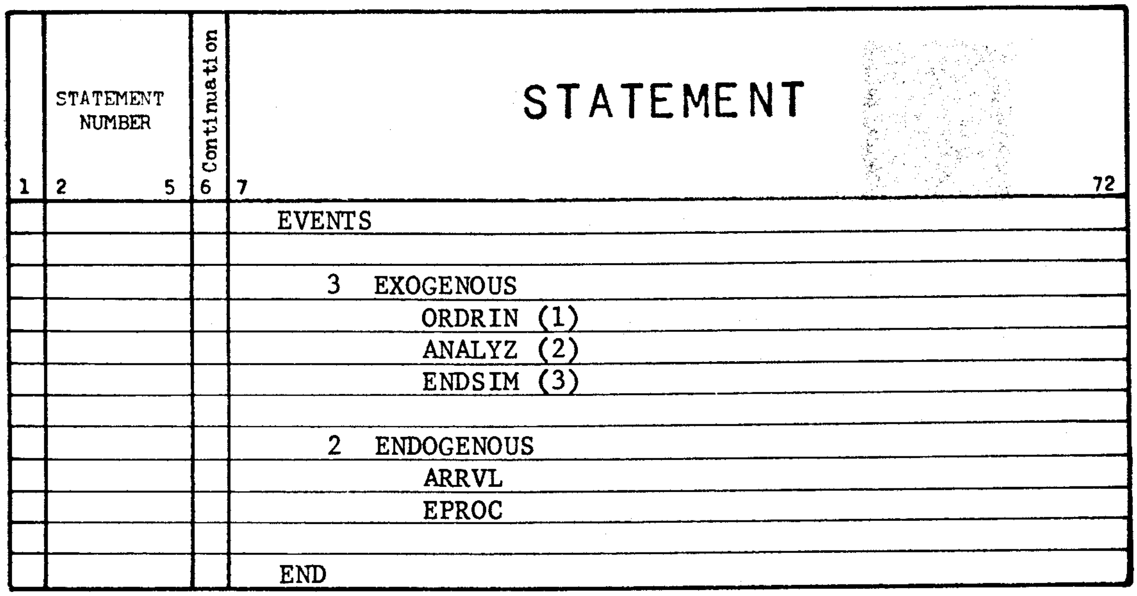
In SIMSCRIPT, Function subprograms may or may not have arguments. Only Local Variables may appear as arguments in a FUNCTION statement, but when the Function is mentioned in other routines, the arguments may be expressions containing Local and System Variables.

Control is returned to the routine calling for the Function by means of the RETURN statement. An END statement appears at the physical end of each Function subprogram.

In addition to written Function subprograms, a number of other Function subprograms are automatically generated by the translator. These subprograms are discussed in Chapter 11.

EVENTS LIST

For the translator to generate an appropriate Timing routine, it is necessary to list each kind of Exogenous and Endogenous Event in what is called an Events List. The Events List may be written on a standard coding sheet as shown in Fig. 13.



1
STATEMENT NUMBER
2
5
Continuation
6
7
STATEMENT
72

```
EVENTS

    3  EXOGENOUS
            ORDRIN (1)
            ANALYZ (2)
            ENDSIM (3)

    2  ENDOGENOUS
            ARRVL
            EPROC

END
```

Fig. 13 — Example of an Events List

The first line of the Events List consists of the word "EVENTS."

GENERAL FORM	EXAMPLES
EVENTS	EVENTS

The next line contains the word "EXOGENOUS" or "EXOG" preceded by the number of different kinds of Exogenous Events to be included in the Timing routine. The name of each type of Exogenous Event followed by a unique identification number enclosed in parentheses is indicated next, with a separate line for each kind of Exogenous Event. The identification numbers need not be sequential or presented in order; they are used to identify the various kinds of Exogenous Events on the Exogenous Event Tape.

Following the definition of the Exogenous Events, the number of different kinds of Endogenous Events is indicated, followed by the word ENDOGENOUS or ENDOG. The name of each kind of Endogenous Event is then listed on a separate line. Endogenous Events do not have identification numbers. If desired, the definition of Endogenous Events may precede the definition of Exogenous Events in the Events List.

The Events List is terminated by an END statement.

SYSTEM PACKAGE

Each SIMSCRIPT object program must contain a translator generated subprogram referred to as the System Package. The System Package is different for simulation and non-simulation programs, but otherwise does not vary from program to program.

A simulation or non–simulation System Package may be obtained by inserting a card containing the word SIMULATION or NON–SIMULATION in the Compile Deck as described in Chapter 12.

GENERAL FORM	EXAMPLES
SIMULATION or: NON-SIMULATION	SIMULATION NON-SIMULATION

The word SIMULATION or NON–SIMULATION must appear somewhere in Cols. 7 through 72, and the card is included in the Compile Deck as though it were a source language subprogram.

ENTITY, ATTRIBUTE, AND SET PACKAGES

A separate package of subprograms is automatically generated for each Entity, Attribute, and Set defined on the Definition Form. These packages are described below.

For each type of Temporary Entity and each type of Event Notice, there is a package containing the subprograms that create and destroy their Attribute records. These subprograms are called on only by the CREATE and DESTROY statements.

For each Permanent Entity, there is a short subprogram for retrieving the value of the automatically defined System Attribute indicating the number of Entities of that particular type. This subprogram is called on by mentioning the System Attribute "NEntity name" in any statement. It is also automatically called on by the FOR EACH "ENTITY" statement.

For each kind of Attribute defined on the Definition Form, whether Temporary or Permanent, there is a generated package of subprograms for storing and retrieving its values. These subprograms are called on as Attributes are mentioned in the various source language statements.

For each kind of Set, there is a package of subprograms·which accomplishes the operations involved in the FILE and REMOVE FIRST statements. If the Set is a Ranked Set, the package also includes a subprogram for accomplishing the operations of the REMOVE "SPECIFIC" statement.

MAIN ROUTINE

If a non–simulation program is to be written in SIMSCRIPT, the Events List and the resulting Timing Routine are omitted. A Control routine must be written in place of the Timing Routine, and it must begin with the statement "MAIN."

GENERAL FORM	EXAMPLES
MAIN "optional text"	MAIN ROUTINE MAIN

This control routine will be the first routine executed following the input of the Initial Conditions Deck described in Chapters 12 and 14.

If a "MAIN" routine is written, a non–simulation System Package must be used. The "MAIN" routine must end with an END statement.

Chapter 10

REPORT GENERATOR

The Report Generator generates output routines based on the contents of the Report Generator Layout Form shown in Fig. 14. As many different Report subprograms as are desired may be included in a particular program.

As can be seen in Fig. 14, the Layout Form contains a number of control columns and 131 print positions. Each line of the Layout Form is punched into a pair of cards. A double line separates the right– and left–hand cards, along which the form may be folded for ease in key punching. The order of punching and card presentation consists of all left–hand cards in order, followed by all right–hand cards in order. The horizontal shaded rows have no significance except as a visual aid in distinguishing one row from another. The rules for completing the Layout Form are described below.

REPORT AND END STATEMENTS

Each report must have a unique name, and the first entry on the Layout Form must be a "REPORT" statement written in the print positions of the left–hand card indicating the Report name and arguments, if any. Arguments may be continued onto the right-hand card.

GENERAL FORM	EXAMPLES
REPORT "Report name" ("arguments, if any")	REPORT FINAL REPORT INTER(BASE)

Report subprograms are called in the same manner as ordinary Subroutines, with the CALL statement in the general form, "CALL 'Report name' ('arguments, if any')."

The "Report name" may consist of one to six letters or numbers. The first character must be a letter, and the name may not end in "F" or contain "XX" or a special character.

Argument values are given to a Report by the calling routine. Arguments mentioned in a REPORT statement must be Local Variables, but expressions containing Local or System Variables may be used as arguments in

PROGRAMMER ____________________

PROBLEM ____________________

DATE ____________________

SIMSCRIPT REPORT GENERATOR LAYOUT FORM

PAGE ____ OF ____

PAGE ____ OF ____

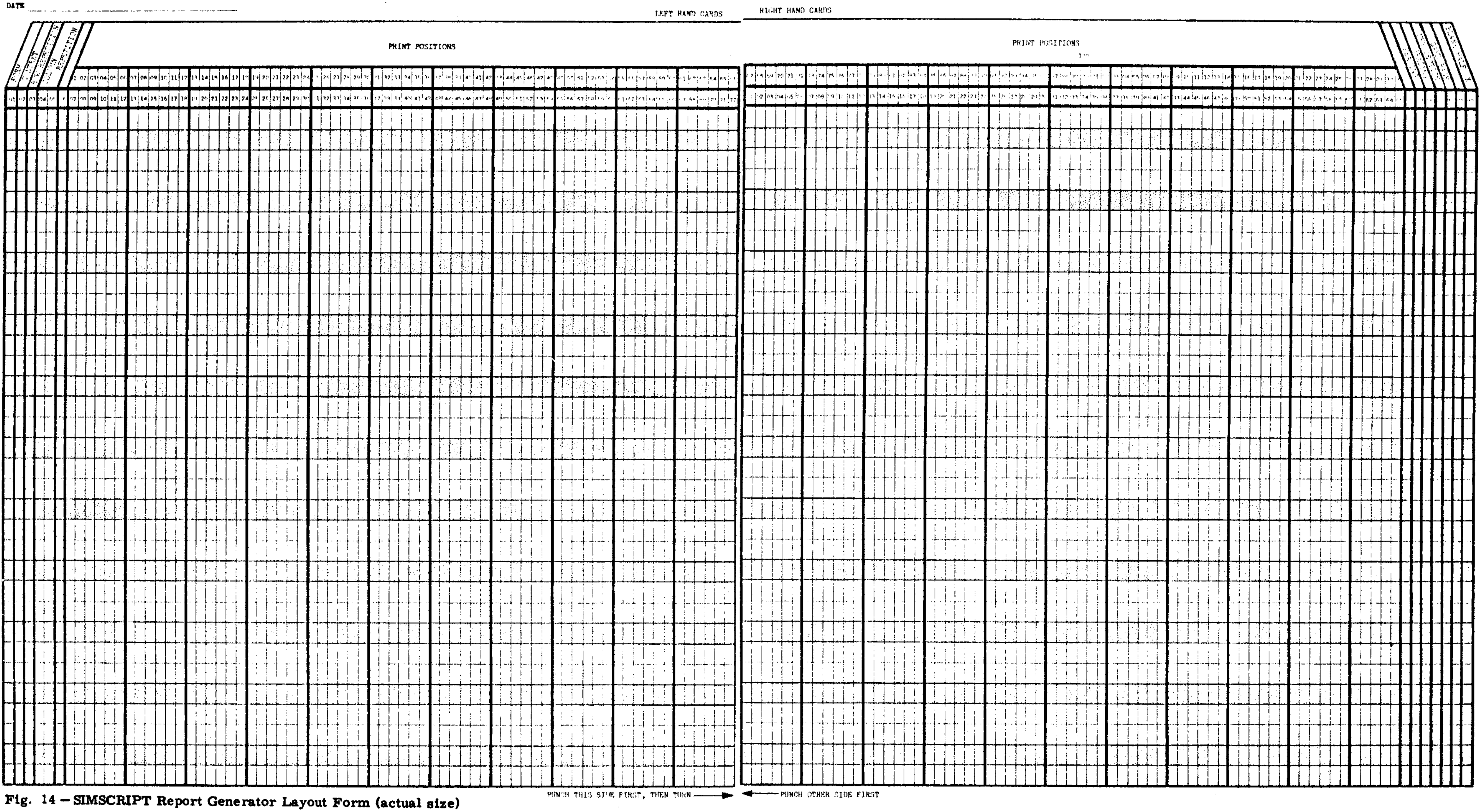


Fig. 14 — SIMSCRIPT Report Generator Layout Form (actual size)

the CALL statement. The last line of each Report description must always consist of the word "END" written in the print positions of both the right–hand and left–hand cards (RETURN statements are not used by the Report Generator since the logical end and physical end are always the same). The line containing the END statement must contain no other entries.

FORM LINES

Print positions for text, integers, decimal numbers, and stored alphanumeric data are assigned by means of Form Lines. A Form Line must contain a mark in the column headed "Form." Print positions are then assigned in the following manner.

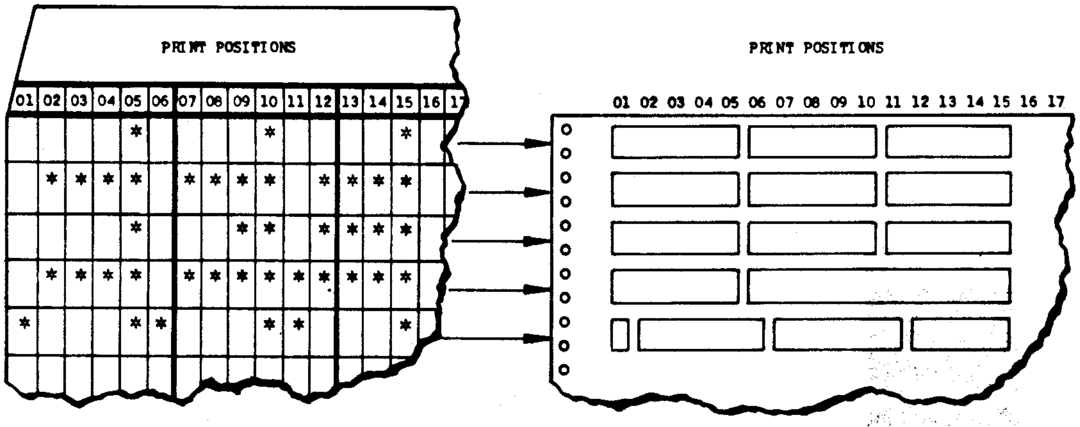
Text

Text is written into the print positions where it is desired to have it appear. Asterisks may never be used in text since they always designate print positions for the value of Variables or Functions, which are to be defined on a "Content Line" following the Form Line.

Integers

An integer field is indicated by an asterisk in the print position where the units position is to go. The field will be as wide as there are unassigned print positions to the left. Although the field can be of any size, 131,071 is the largest value an integer Local or System Variable can take on.

If desired, the asterisk in the units position of an integer field may be preceded by additional asterisks to help in visualizing the appearance of the printed output. Such asterisks must be continuous, however, since a blank would be interpreted as indicating the start of another field. Integer field layout conventions are illustrated in the following examples.



PRINT POSITIONS

01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17
				*					*					*		
	*	*	*	*		*	*	*	*		*	*	*	*		
				*				*	*		*	*	*	*		
	*	*	*	*		*	*	*	*	*	*	*	*	*		
*				*	*				*	*				*		

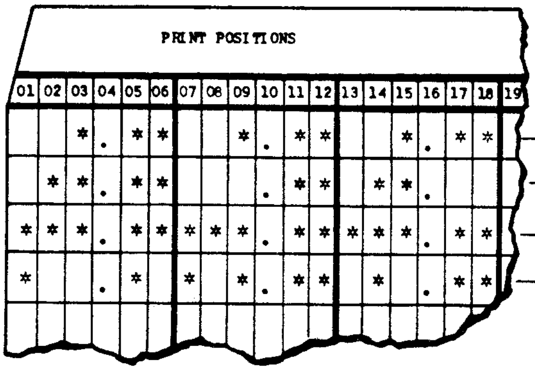
Integer Fields on Layout Form   Size and Location of Printed Integer Fields

The smallest possible successive integer field is two positions wide, since at least one space and one asterisk are required to define a separate field. One–digit integer fields are possible, however, if separated by text.

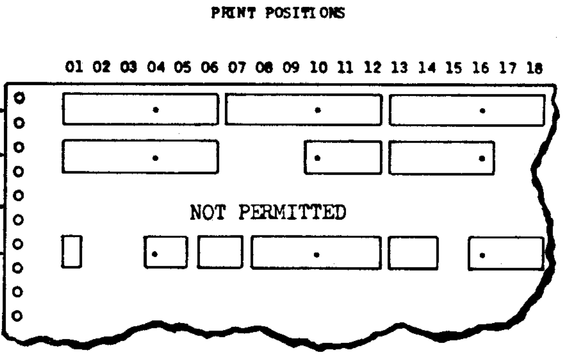
Leading zeros are automatically suppressed.

Decimal Numbers

A decimal field is indicated by a decimal point in the desired print position, asterisks in all decimal places to the right of the decimal point, and an asterisk in the units position to the left of the decimal point. Additional asterisks may be inserted preceding the units position to help visualize the layout, so long as at least one space is left to separate successive fields. If no places are specified to the right of the decimal point, only the truncated integer portion and decimal point will be printed. Similarly, if the asterisk is omitted from the units position to the left of the decimal point, only the decimal point and fractional part will be printed. Conventions for specifying decimal fields are illustrated in the following examples.



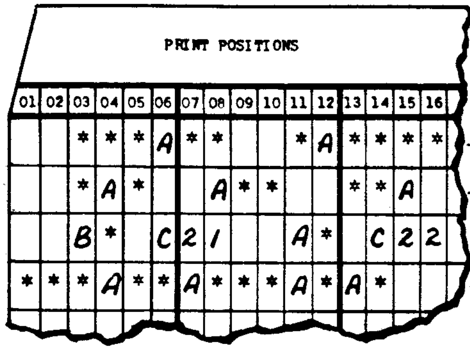
Decimal Fields on Layout Form



Size and Location of Printed Decimal Fields

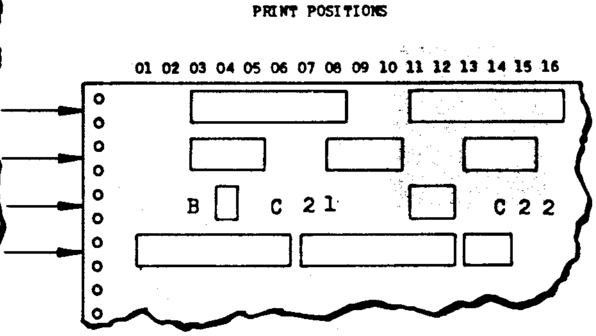
Stored Alpha–Numeric Data

Print positions for stored alpha–numeric data are assigned by placing an asterisk in each print position in the field, but the letter "A" must appear at least once instead of an asterisk. There is a six–character maximum for the alpha–numeric field per Variable specified in the accompanying Content Line. The minimum is two characters, since the letter "A" without an adjoining asterisk is interpreted as text. Alpha–numeric fields may be written side by side without intervening spaces. A continuous series of alpha–numeric fields will always be interpreted from left to right as a series of six–character fields, except that the last field will contain less than six characters if the total number of assigned print positions is not an even multiple of six.



PRINT POSITIONS 01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
		*	*	*	A	*	*			*	A	*	*	*	*
		*	A	*			A	*	*			*	*	A	
		B	*		C	2	/			A	*		C	2	2
*	*	*	A	*	*	A	*	*	*	A	*	A	*		

Alpha-numeric Fields on Layout Form



Size and Location of Printed Fields

If desired, two or more field descriptions may be written side by side without intervening spaces provided they are not ambiguous. Possible ambiguities are "*AB", "A.*", "*..", "*.C".

Every Form Line that contains one or more asterisk fields for integer, decimal, or stored alpha-numeric data must be followed by a Content Line identifying either Variables or Functions whose values are to be printed in the indicated fields.

CONTENT LINE

Each Content Line is marked with a character in the column headed "Content." One Variable or Function must be specified in the Content Line for each field of asterisks in the preceding Form Line. Variables may be Local or System and may or may not be subscripted (the only Local Variables that could occur would be the arguments in the REPORT statement, or Local Variables in FOR or FOR EACH "ENTITY" control phrases). Functions may be written subprograms or may be Functions automatically generated by the SIMSCRIPT translator or by the FORTRAN Library Tape. Variables and Functions correspond in order with the preceding field of asterisks. Successive Variables and Functions must be separated by at least one space or comma. Although it is not necessary, it is helpful to write each Variable or Function directly under its respective asterisk field if space permits.

One Content Line for each Form Line is usually sufficient for defining the Variables and Functions whose values are to be printed. This is not necessarily true, however, since Variable names and subscripts can be longer than the fields in which their values are to be printed. If required, a Content Line may be continued on the next line by placing a mark in Col. 2 of each continuing line. The name of a Variable or Function may not be split between the end of one line and the beginning of the next.

HEADING CODES

Heading codes designate on which of successive pages of a Report Section a Form Line is to be printed. These codes are:

"Blank" in the column entitled "Heading" will print the Form Line once on each page of the Report Section.

"1" under "Heading" will print the Form Line once on the first page only of the Section.

"2" under "Heading" will print the Form Line once on all but the first page of the Section.

"3" under "Heading" will print the Form Line on every page until the control phrases in a subsequent Row Repetition Line are satisfied.

SPACING CODES

Spacing codes indicate how many lines are to be skipped following the printing of a Form Line. A number from "1" to "9" is inserted in the column entitled "Spacing, " depending on how many lines are to be skipped following each Form Line. If no lines are to be skipped, zero is inserted or the column is left blank.

The number of lines per page of printed output is an automatically defined System Attribute called "LINES" and is always set equal to 55 by the translator (see Chapter 11). If a different number is desired, the value of the Attribute "LINES" may be specified on the System Specification Card described in Chapter 12, or it may be modified by a source language statement prior to calling the REPORT Routine.

NEW SECTION CODE

A given Report may be divided into two or more Sections. The start of a new Section is indicated by placing a mark in the column entitled "New Section" on the first line of the new Section. Starting a new Section in effect starts a new Report without control having been returned to the calling routine. Unless suppressed by a Same Page code, the first line of each new Section will start on a new page.

SAME PAGE CODE

If it is desired to continue a new Report on the same page as the end of the previous Report, a mark should be placed under "Same Page" on the line following the REPORT statement. If it is desired to continue a new Section on the same page as a previous Section; insert a mark under "Same Page" on the line containing the New Section Code.

BLANK HALF

For every left–hand card there must be a corresponding right–hand card in order for the translator to generate the REPORT Routine. If a line has no other entries on the right–hand card, a mark is inserted on the right–hand card under "Blank Half" to insure that it will be punched and included in the deck.

PAGE NUMBERS

For convenience in numbering pages, there is an automatically defined System Attribute called "PAGE" (see Chapter 11). This Attribute is automatically set equal to "1" at the start of object program execution and is advanced by "1" each time a page of output is completed. It may be modified at any time by source language statements, and its value may be printed like that of any other integer Variable by means of a pair of Form and Content entries.

ROW REPETITION

If it is necessary to output a table with a variable number of rows, this can be accomplished with a Row Repetition Line. Place a mark under "Row Repetition" on the line immediately following the Form and Content Lines describing the columns of the table. One or more FOR, FOR EACH "ENTITY" or FOR EACH OF "SET" control phrases, perhaps modified by WITH, OR or AND phrases may be inserted in the Row Repetition Line to control the printing of the Variables and Functions specified in the Content Line. Use of Row Repetition was illustrated in the examples in Chapters 1 and 3(see Figs. 2 and 10).

Only one series of control phrases for Row Repetition is permitted in each Report Section. If necessary, these control phrases may be continued on successive lines of the Layout Form by placing additional marks under "Row Repetition."

If successive pages are required to print the number of rows specified by the control phrases, headings will be repeated as specified by the previous Heading and Spacing Codes within the particular Report Section.

COLUMN REPETITION

A Column Repetition Line can be used if it is desired to print a series of subscripted quantities across a row where the number of different values the subscript can take on may vary. The main use of this feature is in printing tables where both the number of rows and the number of columns are either specified separately for each simulation run or are determined as part of the simulation process. Column Repetition also permits the printing of a list across a row instead of down a column.

When Column Repetition is to be used, the "Column Repetition Line" must be the first line following the REPORT statement, or the first line of a new Report Section. Only one Column Repetition Line is permitted in each Report Section, and it remains in effect throughout the Section.

Figure 15 depicts the use of both Column and Row Repetition for printing a two–dimensional table with 24 table columns per page. For purposes of discussion, the various lines are numbered in the left margin of the figure.

The second line in Fig. 15 is a "Column Repetition Line," and is so indicated by inserting the number of cases to be printed across each page in columns entitled "Column Repetition." The control phrase "FOR EACH DEPOT J" specifies the cases for which Variables are to be repeated across the rows of the Report. One or more FOR, FOR EACH "ENTITY," or FOR EACH OF "SET" phrases, optionally followed by any number of WITH, OR, or AND phrases, may be inserted in a Column Repetition Line. The control phrases may be written across both the left– and right–hand cards, and they may be continued on following lines by inserting additional non–zero marks under "Column Repetition."

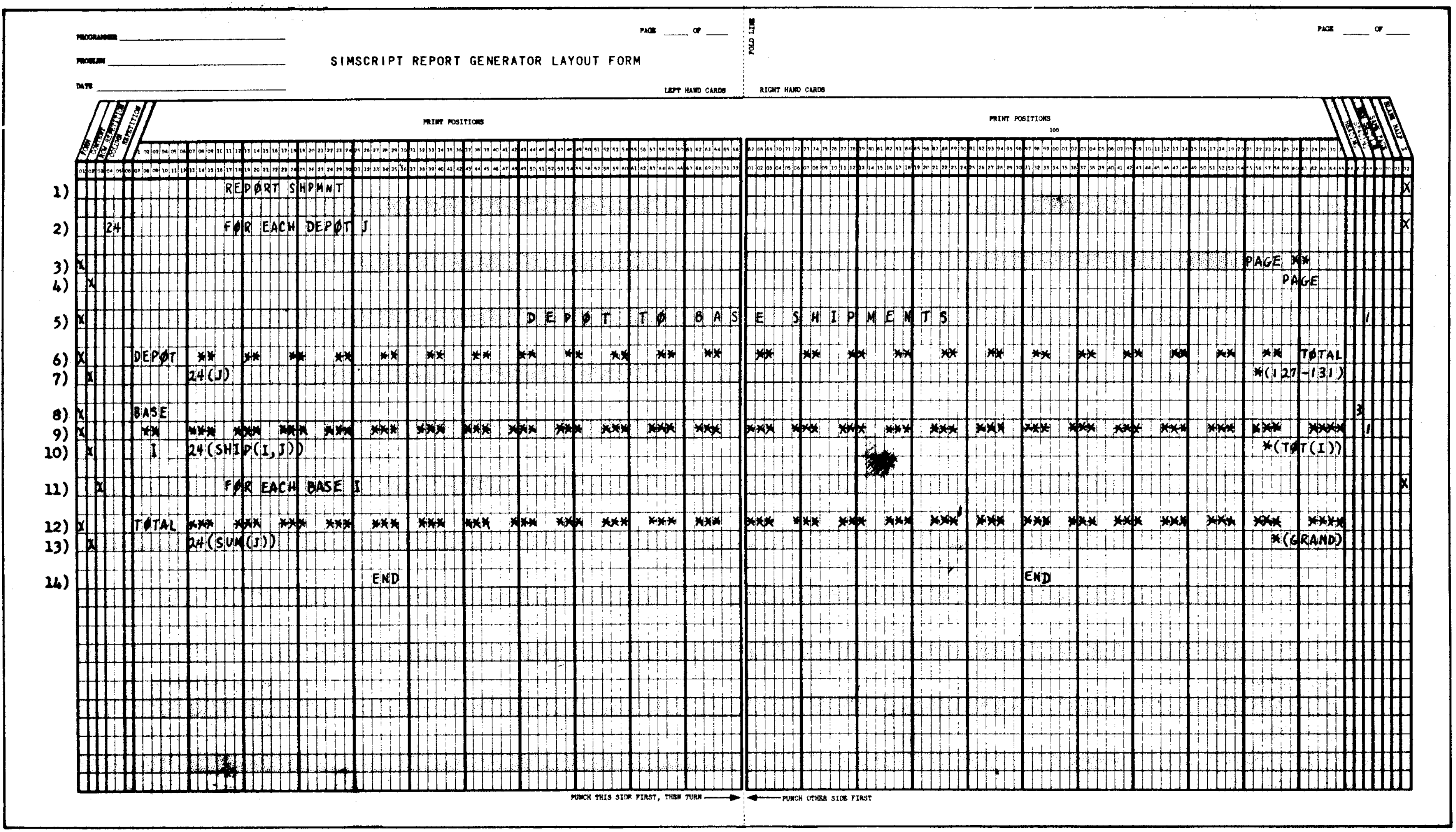
When WITH, OR, or AND phrases are used to impose conditions on the columns of a table, care must be taken that these conditions are independent of the control phrases in the subsequent Row Repetition Line, and vice versa. In other words, WITH, OR, or AND phrases may be used to delete entire columns or entire rows from the printed output, but not to delete a particular cell at the intersection of a row and column.

Lines 3 and 4 are Form and Content Lines for printing page numbers (the Variable "PAGE" is the previously mentioned translator–defined System Attribute).

Line 5 is a Form Line containing the text "DEPOT TO BASE SHIPMENTS," which will be printed on every page to the end of the Report.

Line 6 is a Form Line containing the text "DEPOT," followed by 24 integer fields for printing the identification number of each Depot. The text word "TOTAL" appears at the end of the line. Since the Heading code is blank, this Form Line will be printed on each page of the Report.

The accompanying Content Line (line 7) contains two entries. The first, "24(J)," specifies that successive values of the Local Variable "J" are to be printed in the 24 integer print fields defined on the preceding Form



PROGRAMMER ____________ PAGE ____ OF ____

PROBLEM ____________

SIMSCRIPT REPORT GENERATOR LAYOUT FORM

DATE ____________

LEFT HAND CARDS | FOLD LINE | RIGHT HAND CARDS PAGE ____ OF ____

FORM
CONTENT
ROW REPETITION
COLUMN REPETITION
PRINT POSITIONS
PRINT POSITIONS
100
SPACING
SKIP PAGE
BLANK HALF

```
1)                         REPØRT SHPMNT                                                                  X
2)      24                 FØR EACH DEPØT J                                                               X
3)  X                                                                                  PAGE **
4)    X                                                                                    PAGE
5)  X                                     D E P Ø T   T Ø   8 A S E   S H I P M E N T S                 1
6)  X        DEPØT  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  **  TØTAL
7)    X             24(J)                                                                  *(127-131)
8)  X        BASE                                                                                      3
9)  X          **   ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ****   1
10)   X         I   24(SHIP(I,J))                                                          *(TØT(I))
11)     X                  FØR EACH BASE I                                                                X
12) X        TØTAL ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ***  ****
13)   X             24(SUM(J))                                                             *(GRAND)
14)                              END                                          END
```

PUNCH THIS SIDE FIRST, THEN TURN → ← PUNCH OTHER SIDE FIRST

Fig. 15 — Example of the Use of Column Repetition

Line. The general form for specifying the Content for column repetitions is as follows.

GENERAL FORM	EXAMPLES

"integer" ("any number of Variables separated by commas")	12(JOB(QUE))
	4(SHIP(I,J),ATM(I,J))

The "integer" specifies the maximum number of cases to be printed across the page. This value must be the same as previously specified in the Column Repetition Line. Cases exceeding this maximum will be printed on subsequent pages. The other entry in line 7, "*(127–131)," suppresses the text word "TOTAL" until after the control phrase "FOR EACH DEPOT J" is satisfied. If there are more than 24 Depots, the word "TOTAL" will be suppressed on all pages except those containing the final column of the table. This suppression of print fields of the preceding Form Line may be done only when the Form and Content Lines are under the control of a Column Repetition Line, and it may be applied only to print fields to the right of the columns being repeated. The general form of the suppress–notation is:

GENERAL FORM	EXAMPLES
*("any number of Variables or print field designations separated by commas")	*(PAP(J)) *(AB(X),121-127,SUM(X))

As shown in the examples, print fields to be suppressed are specified by writing the numbers of the first and last print positions in the field. The two print position numbers are separated by a dash or minus sign.

Line 8 is a Form Line inserting the text word "BASE" as a heading for the first column of the table. It contains a Heading Code "3" so that it will not be repeated beyond the completion of the table rows.

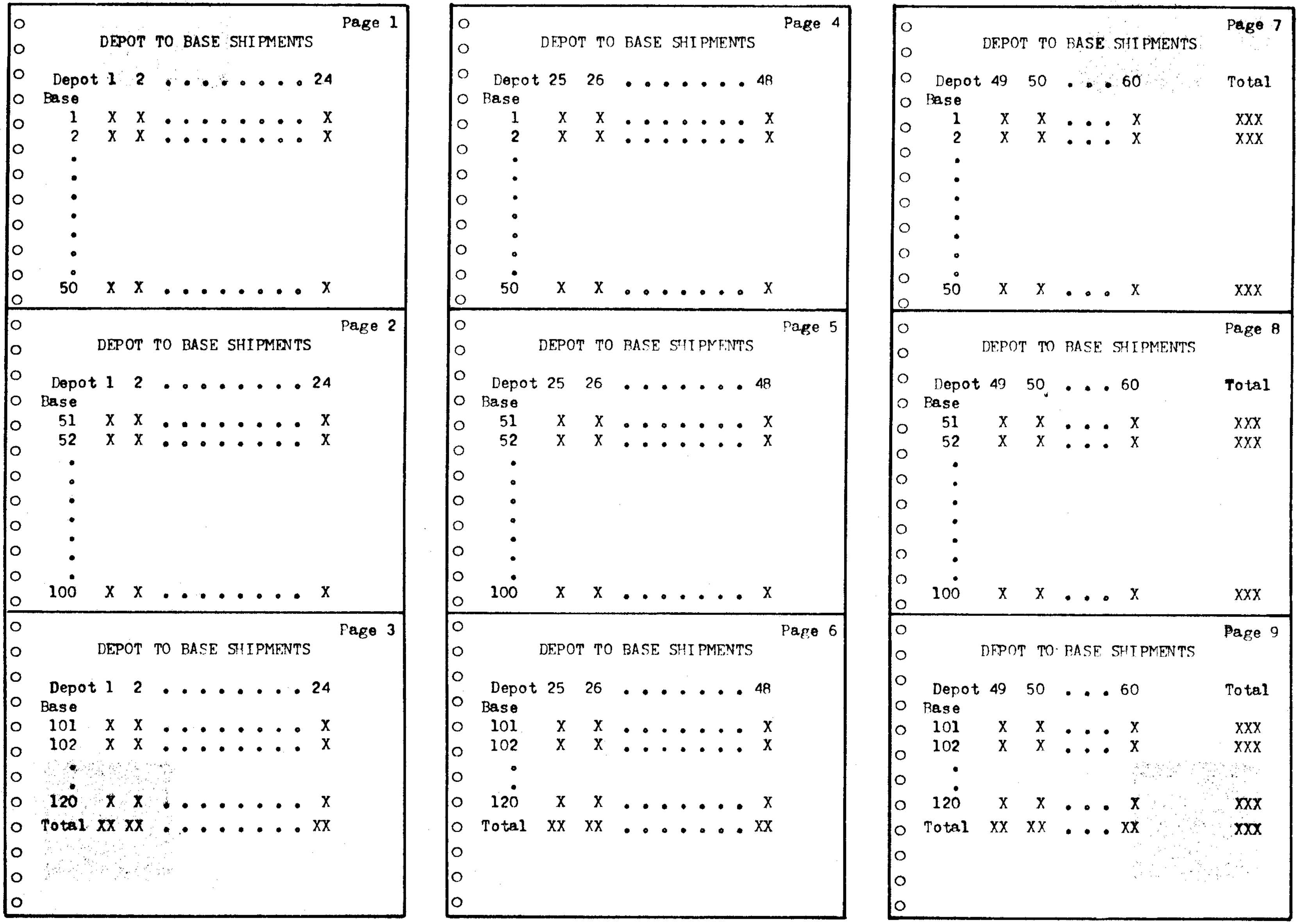
The next pair of Form and Content Lines (lines 9 and 10) define the contents of the table itself. The first integer field is to contain the value of the Local Variable "I" indicating the number of the Base. The next 24 integer fields are to contain the up–to–24 values of "SHIP(I, J)". The final field is for printing the row total "TOT(I)" at the end of the row. If there are more than 24 Depots, the last field is suppressed until the row is completed.

Line 11 is a Row Repetition Line indicating that the output described by the preceding pair of Form and Content Lines is to be repeated "FOR EACH BASE I". One or more FOR, FOR EACH "ENTITY", or FOR EACH OF "SET" phrases, perhaps modified by WITH, OR, or AND phrases, may be inserted in a Row Repetition Line so long as the conditions are independent of the conditions specified in the previous Column Repetition Line.

The final pair of Form and Content Lines (lines 12 and 13) call for printing of column totals and a grand total at the end of the table.

To illustrate the conventions for the continuation of rows and columns on subsequent pages and the suppression of marginal totals, suppose that the Report described in Fig. 15 were to be printed for a case with 60 Depots and 120 Bases. Assuming the standard 55 lines per page, this Report would require nine pages and would be printed as shown schematically in Fig. 16.

The previous example had the values of a single Variable printed across a row; however, the values of any number of Variables can be repeated for successive cases across a row as illustrated in Fig. 17.



```
                                          Page 1
         DEPOT TO BASE SHIPMENTS

   Depot 1  2  . . . . . . . . . 24
  Base
     1   X  X  . . . . . . . . .  X
     2   X  X  . . . . . . . . .  X
     .
     .
     .
     .
     .
     .
     .
    50   X  X  . . . . . . . . .  X
```

```
                                          Page 2
         DEPOT TO BASE SHIPMENTS

   Depot 1  2  . . . . . . . . . 24
  Base
    51   X  X  . . . . . . . . .  X
    52   X  X  . . . . . . . . .  X
     .
     .
     .
     .
     .
     .
     .
   100   X  X  . . . . . . . . .  X
```

```
                                          Page 3
         DEPOT TO BASE SHIPMENTS

   Depot 1  2  . . . . . . . . . 24
  Base
   101   X  X  . . . . . . . . .  X
   102   X  X  . . . . . . . . .  X
     .
     .
   120   X  X  . . . . . . . . .  X
  Total XX XX  . . . . . . . . . XX
```

```
                                          Page 4
         DEPOT TO BASE SHIPMENTS

   Depot 25   26   . . . . . . . . 48
  Base
     1     X    X  . . . . . . . .  X
     2     X    X  . . . . . . . .  X
     .
     .
     .
     .
     .
     .
     .
    50     X    X  . . . . . . . .  X
```

```
                                          Page 5
         DEPOT TO BASE SHIPMENTS

   Depot 25   26   . . . . . . . . 48
  Base
    51     X    X  . . . . . . . .  X
    52     X    X  . . . . . . . .  X
     .
     .
     .
     .
     .
     .
     .
   100     X    X  . . . . . . . .  X
```

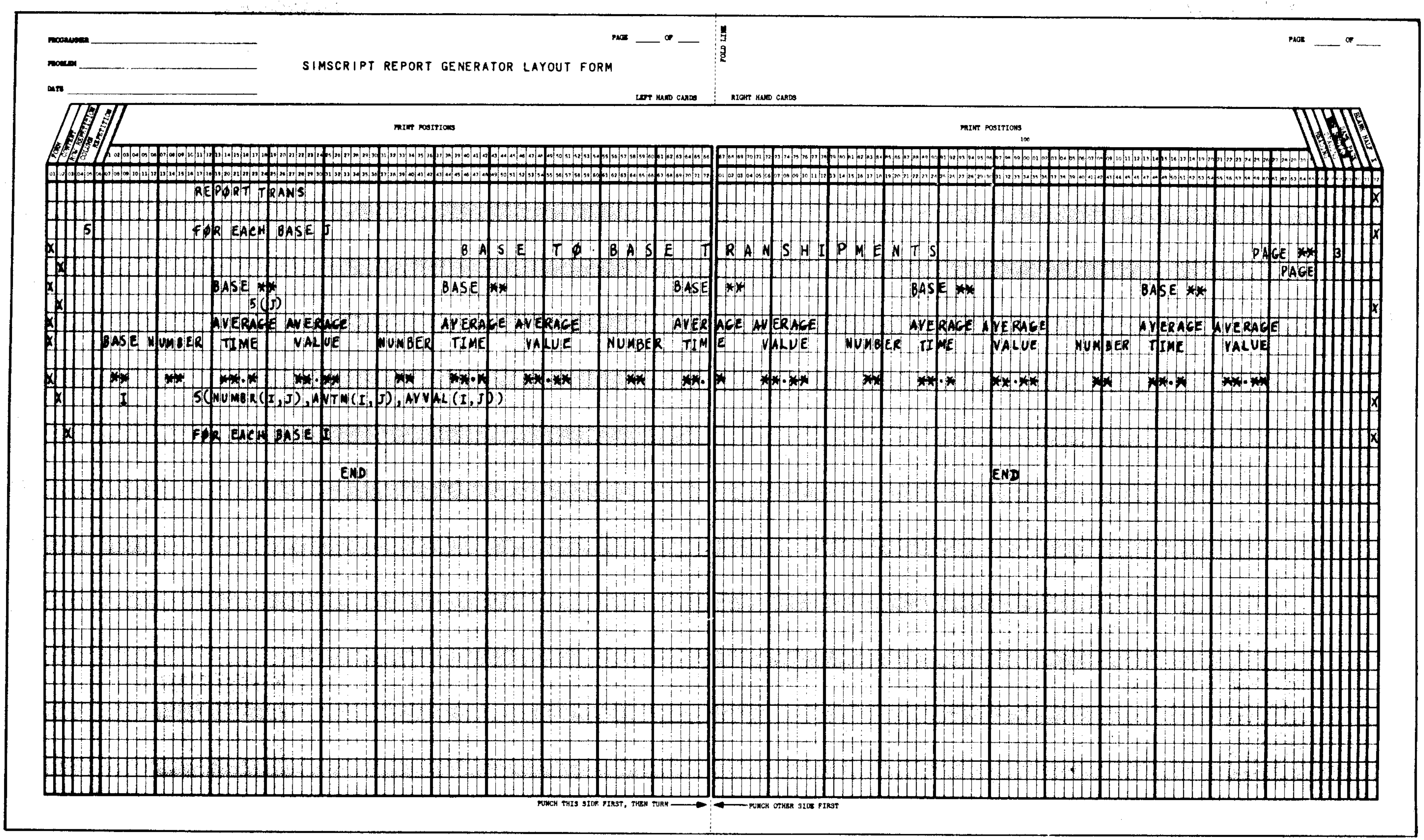
```
                                          Page 6
         DEPOT TO BASE SHIPMENTS

   Depot 25   26   . . . . . . . . 48
  Base
   101     X    X  . . . . . . . .  X
   102     X    X  . . . . . . . .  X
     .
     .
   120     X    X  . . . . . . . .  X
  Total   XX   XX  . . . . . . . . XX
```

```
                                          Page 7
         DEPOT TO BASE SHIPMENTS

   Depot 49   50   . . . 60          Total
  Base
     1     X    X  . . .  X           XXX
     2     X    X  . . .  X           XXX
     .
     .
     .
     .
     .
     .
     .
    50     X    X  . . .  X           XXX
```

```
                                          Page 8
         DEPOT TO BASE SHIPMENTS

   Depot 49   50   . . . 60          Total
  Base
    51     X    X  . . .  X           XXX
    52     X    X  . . .  X           XXX
     .
     .
     .
     .
     .
     .
     .
   100     X    X  . . .  X           XXX
```

```
                                          Page 9
         DEPOT TO BASE SHIPMENTS

   Depot 49   50   . . . 60          Total
  Base
   101     X    X  . . .  X           XXX
   102     X    X  . . .  X           XXX
     .
     .
   120     X    X  . . .  X           XXX
  Total   XX   XX  . . . XX           XXX
```

Fig. 16 – Example of Row and Column Continuation with Suppression of Marginal Totals



PROGRAMMER ______ PROBLEM ______ DATE ______

SIMSCRIPT REPORT GENERATOR LAYOUT FORM

PAGE ____ OF ____ LEFT HAND CARDS | FOLD LINE | RIGHT HAND CARDS PAGE ____ OF ____

FORM / CONTENT / R/W REPETITION / COLUMN REPETITION — PRINT POSITIONS — PRINT POSITIONS 100 — BLANK HALF I

```
                   REPØRT TRANS                                                                                       X

    5              FØR EACH BASE J                                                                                    X
X                                         B A S E   T Ø   B A S E   T R A N S H I P M E N T S             PAGE **  3
  X                                                                                                           PAGE
X                     BASE **                   BASE **                  BASE **                 BASE **                 BASE **
  X                       5(J)                                                                                        X
X                     AVERAGE AVERAGE           AVERAGE AVERAGE          AVERAGE AVERAGE         AVERAGE AVERAGE         AVERAGE AVERAGE
X       BASE NUMBER    TIME     VALUE   NUMBER   TIME     VALUE   NUMBER  TIME     VALUE   NUMBER  TIME    VALUE   NUMBER  TIME     VALUE

X        **    **     **.*     **.**     **      **.*     **.**    **     **.*     **.**    **     **.*     **.**    **     **.*     **.**
  X      I         5(NUMBR(I,J),AVTM(I,J),AVVAL(I,J))                                                                 X

   X               FØR EACH BASE I                                                                                    X

                               END                                                       END
```

PUNCH THIS SIDE FIRST, THEN TURN → | ← PUNCH OTHER SIDE FIRST

Fig. 17 — Column Repetition for Groups of Variables

Chapter 11

AUTOMATICALLY GENERATED SYSTEM VARIABLES AND FUNCTIONS

A number of System Attributes and Functions are automatically defined by the SIMSCRIPT translator, and their related subprograms are always included in the System Package. System Variables representing the dimensions of Permanent Attribute lists and tables are also automatically defined, based on the contents of the Definition Form. If desired, subprograms are generated for determining the values of Permanent Attributes by Random Look–Up Table procedures. These Variables and Functions are discussed here under the following headings:

Time Variables and Functions
Output Variables
Random Variables and Functions
Random Look–Up Tables
List and Table Dimensions
FORTRAN Library Routines and Functions

TIME VARIABLES AND FUNCTIONS

During a simulation run, simulated time is accounted for in days and fractions of days. Time values to be read in as input may be expressed in other modes, and will be automatically converted to decimal–days. Three automatically defined System Attributes — "TIME," "HOURS," and "MINS"— are related to the method of accounting for simulated time and the conversion of input time values to a common mode. To permit the output of time values in various modes, four Functions — "DECHR," "DPART," "HPART," and "MPART" — are automatically provided as part of the System Package.

"TIME"

This translator–defined floating point System Attribute is always equal to the current value of simulated time expressed in days and fractions of days. It is automatically set equal to zero at the start of simulation. Before an Exogenous Event Routine is called, "TIME" is set equal to the time–value specified for the Event occurrence on the Exogenous Event Card. (The day, hour, and minute specified on the Event Card are automatically converted to days and fractions of days.) Before an Endogenous Event Routine is called, TIME is set equal to the time–value specified in the prior CAUSE statement. The System Variable TIME is available to all subprograms, but its value may not be modified by source language statements. It is not to be defined on the Definition Form.

"HOURS" and "MINS"

These automatically defined floating point System Attributes are equal respectively to the number of hours per simulated day, and of minutes per simulated hour. Prior to object–program execution, they may be assigned any value on the System Specifications Card as described in Chapter 12. During program execution, the values of either or both may be modified at any time by source language statements. If the values of HOURS and MINS are not specified on the System Specifications Card, they are automatically set equal to 24.0 and 60.0 respectively.

"DECHR"

The Simulation System Package contains a Function for converting decimal–days into decimal–hours. The Function has one argument, which may be any floating point expression. The value of the Function is referred to in source language statements as "DECHR(floating point expression)"; it is equal to the floating point expression multiplied by the System Attribute HOURS.

"DPART," "HPART" and MPART"

To convert decimal–days into days, hours, and minutes, integer Functions are provided for determining the "days part," "hours part," or "minutes part" of any time value expressed in decimal–days.

The "days part" Function, referred to as "DPART (floating point expression)," is equal to the integer number of days contained in the value of the floating point decimal–days time expression.

The "hours part" Function, referred to as "HPART (floating point expression)," is equal to the integer number of hours contained in the remainder of the time expression after the "days part" has been subtracted from it.

The "minutes part" Function, referred to as "MPART" (floating point expression)," is equal to the integer number of minutes in the remainder of the time expression after the days and hours parts have been subtracted from it.

The HPART and MPART conversions are based on the current values of the System Attributes HOURS and MINS.

OUTPUT VARIABLES

Three translator–defined integer System Attributes — "PAGE," "LINES," and "OTAPE" — are related to the use of the Report Generator.

"PAGE"

"PAGE" is an integer System Attribute set equal to one at the start of object–program execution and automatically advanced by one every time a page of printed output is completed.

The value of PAGE may be modified at any time by source language statements.

"LINES"

The value of the System Attribute "LINES" specifies the number of lines per page of printed output; it is automatically set equal to 55. If a different number of lines is desired, the value of LINES may be specified on the System Specification Card or may be modified by source language statements up to a maximum of 60 lines.

"OTAPE"

The value of the System Attribute "OTAPE" specifies the logical tape unit on which Report Generator output is to be written. The value of OTAPE is automatically set equal to six. If a different tape is to be used, it may be specified under "Report Tape" on the Systems Specifications Card or the Attribute OTAPE may be modified at any time by source language statements.

RANDOM VARIABLES AND FUNCTIONS

Because of the frequent use of random variables in simulation problems, SIMSCRIPT supplies rectangularly distributed integer and floating point random numbers. One translator—defined Function, "RANDI(I, J), " and two System Attributes, "RANDM" and "RANDR" are provided for this purpose. Random look—up table procedures are also provided for generating random variables with non—rectangular distributions, as described later in the Chapter.

"RANDI(I, J)"

The successive values of this translator—defined Function are random integers uniformly distributed from "I" to "J." The arguments "I" and "J" may be Local or System integer Variables or integer constants. A new value of RANDI(I, J) is generated every time it appears in the source program, even when it is repeated in the same statement. Its value cannot be modified by source language statements.

"RANDM"

The successive values of "RANDM" are random floating point numbers uniformly distributed from "0.0" to "1.0." A new value of RANDM is generated every time it appears, even when it is repeated in the same statement. The value of RANDM cannot be modified by source language statements.

"RANDR"

The value of "RANDR" is the root used to generate the next value of RANDI(I, J) or RANDM. RANDR is a "right-adjusted" integer variable and may be given any odd initial value at the time of object–program execution. The value of RANDR may be modified at any time by source language statements. Specific values may also be read in using the right adjusted integer field description discussed in Chapter 7. If its initial value is not specified, it is automatically set equal to "1." RANDR may not be an even number.

RANDOM LOOK–UP TABLES

Unsubscripted and single–subscripted Permanent Attributes may, if desired, have their values determined by either of two random look–up procedures. A step function procedure is provided for discrete probability distributions; a linear interpolation procedure is provided for linear approximations to continuous distributions.

Permanent Attributes that are to have their values determined from a random look–up table must be so defined on the Definition Form as described in Chapter 13.

The look–up tables consist of possible Attribute values with their corresponding probabilities. One look–up table is required for each unsubscripted random Attribute. Single–subscripted random Attributes require a series of look–up tables, one for each value the subscript can take on.

The look–up tables themselves are input at the time of object–program execution as part of the Initial Conditions Deck. They may therefore be changed from one run to the next.

Step Function

If the look–up table describes a discrete probability distribution, such as the example shown in Fig. 18, the "step function" procedure must be specified on both the Definition and Initialization Forms as described in Chapters 13 and 14.

The possible Attribute values may be expressed by using any of the FORMAT statement Field Descriptions described in Chapter 7 (except alpha–numeric), and may be given any desired value.

In the case of the step function, the probabilities may be read in as either cumulative or individual probabilities. If individual probabilities are used,

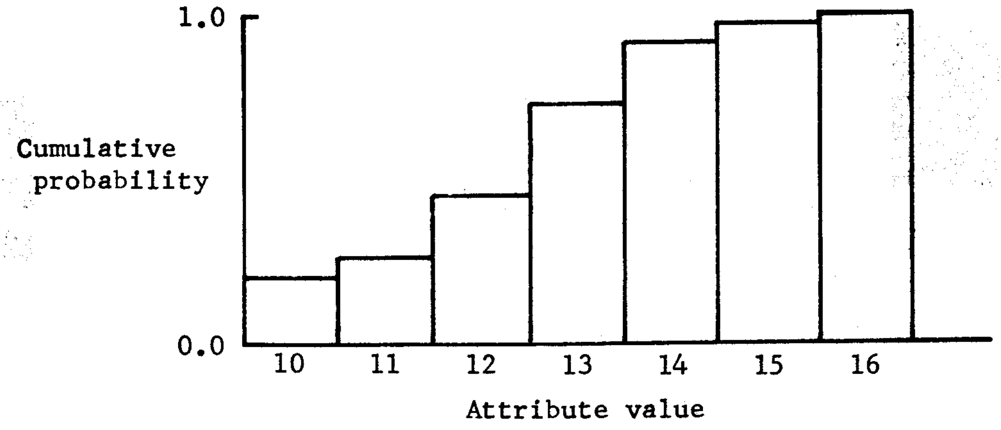


Fig. 18 – Discrete Probability Distribution

they are accumulated in the order of their appearance in the Initial Conditions Deck. Cumulative probabilities must appear in the Initial Conditions Deck in ascending order.

The last cumulative probability should equal 1.0, or individual probabilities should sum to 1.0. In any case, the final cumulative probability is automatically set equal to 1.0 regardless of the probability values that are input.

A new value is automatically obtained every time the Attribute appears in the source program, even if it is repeated in the same statement. The value is determined by generating a value of RANDM and searching the table from the proper entry.

Linear Interpolation

If the Attribute value is described by a continuous rather than a discrete probability distribution, a piece–wise linear approximation such as that shown in Fig. 19 may be used.

In this case, the look–up table describes a series of points from the continuous cumulative probability curve. Each time the Attribute appears in the source program, a new value for RANDM is generated and the table is searched for the pair of cumulative probabilities which bracket that value. The value of the Attribute is then determined by interpolation. The cumulative probability curve may be described by as many points as desired.

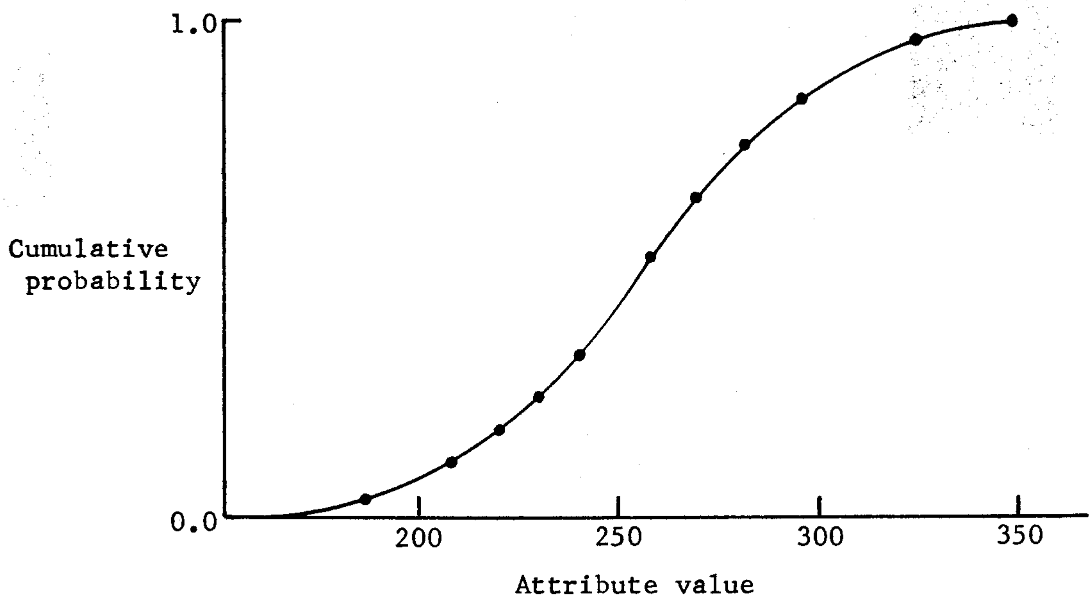


Fig. 19 — Continuous Probability Distribution

For the linear interpolation procedure, the look—up table must be read—in in terms of cumulative probabilities. Furthermore, the cumulative probability of the first possible Attribute value must equal 0. 0; that of the last possible should of course equal 1. 0, and will automatically be set equal to 1. 0 in any case.

One look—up table is required for an unsubscripted Permanent Attribute. If the Attribute is subscripted, one look—up table is required for each value the subscript can take on.

PERMANENT ENTITY AND ATTRIBUTE DIMENSIONS

For each type of Permanent Entity defined on the Definition Form, the SIMSCRIPT translator defines an unsubscripted Permanent Attribute. The name of this automatically defined System Attribute is formed by prefixing the letter "N" to the name of the Permanent Entity Type. For example, if "DEPOT" is a type of Permanent Entity, an unsubscripted Permanent Attribute called "NDEPOT" is automatically defined. It will be noted that "NEntity name" may consist of as many as six characters rather than the usual five. The value of "NEntity name" is specified in the Initial Conditions Deck each time the object program is executed.

For each type of double—subscripted Permanent Attribute, the translator automatically defines a single—subscripted Permanent Attribute called "N'Attribute name'(I)". This variable is equal to the number of entries in the Ith row of the table, and is indispensable when the table is "Ragged," i. e., when a different number of entries can appear in each row.

FORTRAN LIBRARY ROUTINES AND FUNCTIONS

Because SIMSCRIPT is a pretranslator to FORTRAN, all FORTRAN Library Routines and Functions are obtainable from the FORTRAN Library Tape. The arguments used when mentioning these Library Routines may be Local or System Variables.

Table 1

SUMMARY OF AUTOMATICALLY DEFINED VARIABLES AND FUNCTIONS

Name	Value Modifiable by Source Statements	Mode	What the Value Equals
TIME	no	floating point	current value of simulated time in decimal-days
HOURS	yes	floating point	hours per simulated day; unless otherwise specified, it is automatically set = 24.0
MINS	yes	floating point	minutes per simulated hour; unless otherwise specified, automatically set = 60.0
DECHR(arg)	no	floating point argument & floating point result	decimal-hours equivalent of the floating point expression evaluated as decimal-days
DPART(arg)	no	floating point argument & integer result	integer part of floating point expression
HPART(arg)	no	floating point argument & integer result	number of integer-hours if floating point expression were expressed in days, hours and minutes
MPART(arg)	no	floating point argument & integer result	number of integer-minutes if the floating point expression were expressed in days, hours and minutes
PAGE	yes	integer	page no., current page of the printed output
LINES	yes	integer	number of lines per page of printed output; unless otherwise specified, automatically set = 55
OTAPE	yes	integer	number of current output tape for printed reports; unless otherwise specified, it is automatically set = 6
RANDI(I,J)	no	integer arguments & integer results	integer random variable uniformly distributed from "I" to "J" where I and J are integer variables or constants
RANDM	no	floating point	floating point random variable uniformly distributed from 0.0 to 1.0
RANDR	yes	integer	root used to the next random number
NEntity name	no	integer	number of Permanent Entities of the particular type
NAttribute name(arg)	no	integer argument & integer result	number of subscripted values in the indicated row of a Ragged Table

Chapter 12

COMPILATION AND EXECUTION

The compilation and the execution of SIMSCRIPT programs are accomplished through the FORTRAN Monitor. Compilation requires two tapes in addition to the tapes required by FORTRAN. The SIMSCRIPT translator is mounted on tape 12(B6), and tape 11(A6) is used as a SIMSCRIPT scratch tape.

In making the compilation, the SIMSCRIPT translator directly generates FORTRAN object programs for all SIMSCRIPT–provided subprograms, namely, the System Package, the Timing Routine (called "EVENTS"), and the Entity, Attribute, and Set packages described in Chapter 9.

All written SIMSCRIPT source programs (including Report Routines specified on the Report Generator Layout Form) are translated by SIMSCRIPT into FORTRAN source programs, which must then be compiled by the FORTRAN Monitor to obtain FORTRAN object programs. This double processing amounts to two separate FORTRAN jobs. In the following discussion, these jobs are referred to as "Job 1" and "Job 2." In Job 1, SIMSCRIPT source programs are translated to FORTRAN source programs which are written on the scratch tape. In Job 2, the scratch tape is run as another FORTRAN job to obtain FORTRAN object programs, which of course may then be executed in the standard FORTRAN manner without further reference to the SIMSCRIPT translator. The procedure described below accomplishes the translation from SIMSCRIPT into FORTRAN source programs and the subsequent compilation into FORTRAN object programs without getting off of the machine. It also permits SIMSCRIPT jobs to be interspersed among non–SIMSCRIPT jobs on the standard FORTRAN input tape.

COMPILE DECK

The general composition of the Compile Deck is shown in Fig. 20; its various elements are discussed below.

FORTRAN Monitor Type 1 Control Cards

Any FORTRAN Monitor Type 1 Control Cards that may be required by the particular computer installation in order to execute the SIMSCRIPT Loader must appear first in the Compile Deck. These Control Cards all relate to the first job, namely the generation of the SIMSCRIPT–provided object program packages and the translation of written SIMSCRIPT source programs to FORTRAN source programs.

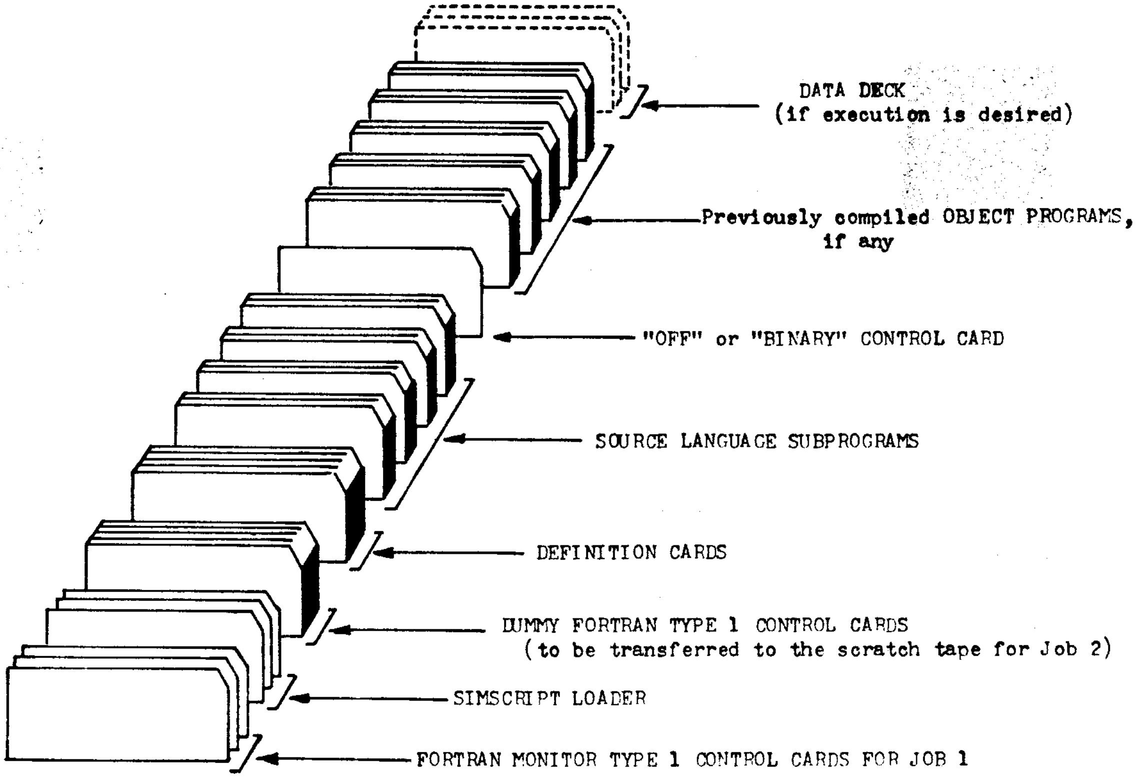

DATA DECK
(if execution is desired)
Previously compiled OBJECT PROGRAMS,
if any
"OFF" or "BINARY" CONTROL CARD
SOURCE LANGUAGE SUBPROGRAMS
DEFINITION CARDS
DUMMY FORTRAN TYPE 1 CONTROL CARDS
(to be transferred to the scratch tape for Job 2)
SIMSCRIPT LOADER
FORTRAN MONITOR TYPE 1 CONTROL CARDS FOR JOB 1

Fig. 20 – Composition of the Compile Deck

SIMSCRIPT Loader

The SIMSCRIPT Loader consists of three cards. The first two cards are a FORTRAN object program that reads in the SIMSCRIPT translator from tape unit 12(B6) and transfers control to the translator. The third card is a FORTRAN Data Control Card (*DATA) since the remainder of the cards in the Compile Deck are, in fact, data as far as Job 1 is concerned.

Dummy Type I Control Cards

These FORTRAN Monitor Type 1 Control Cards relate to the second job, namely the compilation of FORTRAN object programs from the FORTRAN source programs to be generated during the course of Job 1. The translator copies these dummy Type 1 Control Cards onto the scratch tape so that the second job can be run from the scratch tape as a standard FORTRAN job.

Definition Cards

This deck consists of all Definition Cards punched from the SIMSCRIPT Definition Form as described in Chapter 13.

If a Definition Card has appeared in a previous compilation, it must still be included in the deck, but a punch in Col. 72 will suppress the generation of the SIMSCRIPT–provided Entity, Attribute, and Set Packages so that they will not be generated a second time. Each Definition Card must appear with a blank in Col. 72 in at least one compilation, and the appropriate Entity, Attribute, and/or Set Packages (FORTRAN object programs) will be written as output for Job 1 on the standard FORTRAN punch tape (tape 7(B4)). If a change is made in a Definition Card, the changed card is inserted with Col. 72 left blank, and its previously generated subprograms packages must be discarded. The Definition Cards may be in any order.

Source Language Subprograms

The source language subprograms to be compiled follow the Definition Cards in the Compile Deck.* These subprograms may include Exogenous and Endogenous Event routines, Subroutine subprograms, Function subprograms, cards punched from the Report Generator Layout Form, and a "MAIN" routine in the case of a non–simulation program. The Events List and the "SIMULATION" or "NON–SIMULATION" card calling for the System Package are included among these subprograms if they have not already appeared in a previous compilation. Each source language subprogram may, if desired, be preceded by FORTRAN Monitor Type 2 control cards. These control cards will be copied on to the scratch tape so as to be included as part of Job 2.

"OFF" or "BINARY" Control Card

The source language subprograms are immediately followed by an "OFF" Control Card, a "BINARY" Control Card, or the Data Deck, depending on whether it is desired to get off the machine, to load previously compiled object subprograms onto the scratch tape, or to load data on the scratch tape so that all programs may be executed as part of Job 2.

The "OFF" control card, with an asterisk in Col. 1 and the word "OFF" in Cols. 7 through 72, causes an end–of–file mark followed by an end–tape mark to be written on the scratch tape. Control then returns to the FORTRAN Monitor. If another SIMSCRIPT Job 1 is encountered, the translator will erase the end–tape mark on the scratch tape and the input for the new Job 2 will be written following the end–of–file mark for the previous Job 2.

*At least one Definition Card must be included in the Compile Deck even if the subprograms being compiled contain only Local Variables. A "plus" punch in Col. 1 of an otherwise blank card will suffice.

A "BINARY" control card, with an asterisk in Col. 1 and the word "BINARY" in Cols. 7 through 72, causes the object subprograms following it to be transferred to the scratch tape. These object subprograms will be written on the scratch tape and any FORTRAN object programs that may have been generated by the SIMSCRIPT translator during the earlier part of the current Job 1 are also copied from the FORTRAN punch tape onto the scratch tape. All object programs will thus appear on the scratch tape following the SIMSCRIPT–translated FORTRAN source programs, and Job 2 can both compile and execute.

If execution of the object program is desired immediately following the compilation of Job 2, the Data Deck follows the source language subprograms (or the object subprograms, if any) on the Job 1 input tape with no intervening cards. The composition of the Data Deck is described later in the present Chapter.

Previously Compiled Object Subprograms

These subprograms may include both those generated by written SIMSCRIPT or FORTRAN source programs and those automatically generated by the SIMSCRIPT system or by the FORTRAN Library Tape.

EXECUTE DECK

If all subprograms have previously been compiled, the composition of the Execute Deck is as shown in Fig. 21.

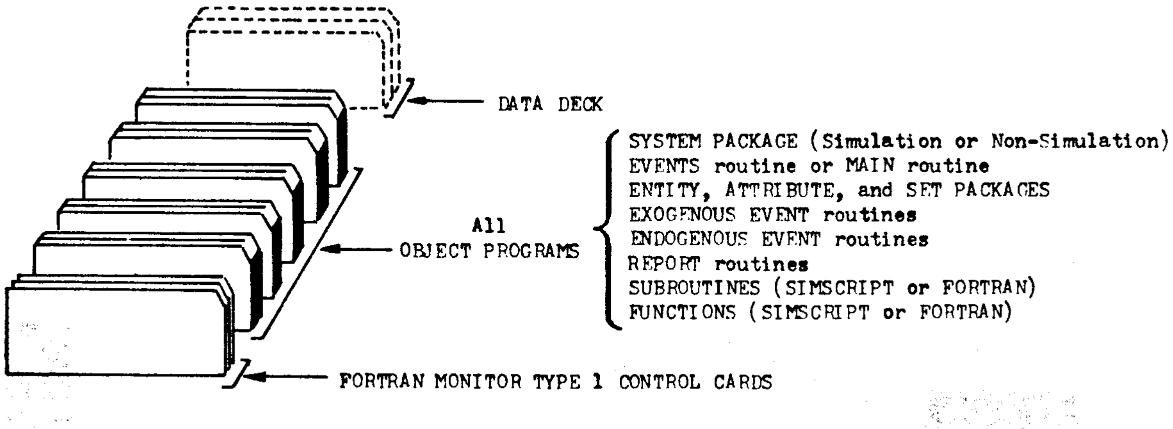

DATA DECK
All
OBJECT PROGRAMS
SYSTEM PACKAGE (Simulation or Non-Simulation)
EVENTS routine or MAIN routine
ENTITY, ATTRIBUTE, and SET PACKAGES
EXOGENOUS EVENT routines
ENDOGENOUS EVENT routines
REPORT routines
SUBROUTINES (SIMSCRIPT or FORTRAN)
FUNCTIONS (SIMSCRIPT or FORTRAN)
FORTRAN MONITOR TYPE 1 CONTROL CARDS

Fig. 21 – Composition of the Execute Deck

FORTRAN Monitor Type 1 Control Cards

The execution of object programs is accomplished in the usual FORTRAN manner, and Type 1 Control Cards as may be required by the particular installation appear first.

Object Subprograms

The object subprograms may appear in any order; they consist of:

A Simulation or Non–Simulation System Package
An EVENTS routine or a MAIN routine
All Entity, Attribute, and Set Packages
All Exogenous Event routines
All Endogenous Event routines
All Report routines
All Subroutines (SIMSCRIPT or FORTRAN).
All Functions (SIMSCRIPT or FORTRAN)

DATA DECK

Figure 22 depicts the composition of the Data Deck. The various elements of the Data Deck are discussed below.

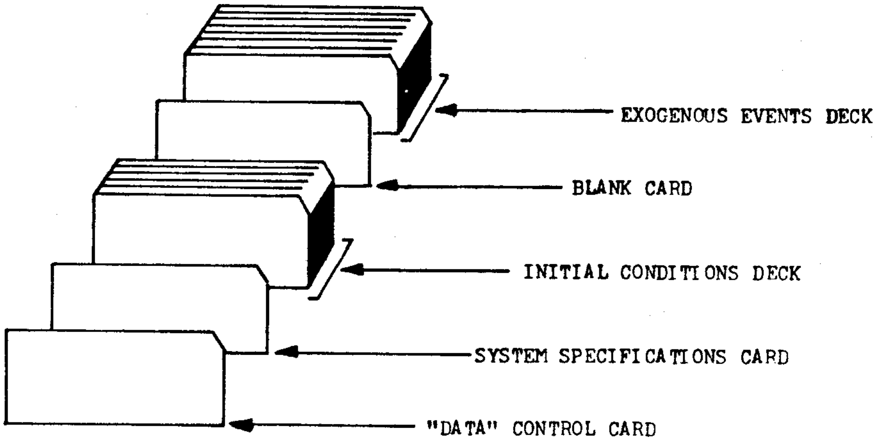


Fig. 22 – Composition of the Data Deck

"DATA" Card

The first card in the Data Deck is a "DATA" control card, with an asterisk in Col. 1 and the word "DATA" in Cols. 7 through 72.

System Specifications Card

The second card in the Data Deck is the System Specifications Card with the following contents and format:

COLUMNS	CONTENTS
1	The number "1".
7 through 12	Maximum "Array Number" that appears in Cols. 32 through 34 of the Definition Form. The units position must be in Col. 12.
13 through 18	The number of minutes per hour of simulated time. It must be an integer with the units position in Col. 18. If this field is left blank, the number of minutes per hour automatically is assumed to be 60. This number may be changed during program execution by modifying the System Attribute MINS.*
19 through 24	The number of hours per day of simulated time. It must be an integer with the units position in Col. 24. If this field is left blank, the number of hours per day is assumed automatically to be 24. This number may be changed during program execution by modifying the System Attribute HOURS.*
25 through 30	The initial value of the root to be used in generating random numbers. If this field is left blank, the initial root is automatically set equal to 1. The root must be an odd number. This root is a System Attribute named RANDR and may also be modified during program execution.
31 through 36	If it is desired to read the Initial Conditions Deck from a tape other than Logical Tape Unit Five, insert the desired tape number in this field. If the field is left blank, the Initial Conditions Deck will be read from Logical Tape Unit Five. The System Specifications Card is always read from Logical Tape Unit Five.

*Although the number of hours per day and the number of minutes per hour are specified as integers on the System Specification Card, the automatically defined System Attributes HOURS and MINS are floating point variables.

37 through 42 If it is desired to read the Exogenous Events Deck from a tape other than Logical Tape Unit Five, insert the desired tape number in this field. If the field is left blank, the Exogenous Events Deck will be read from Logical Tape Unit Five.

43 through 48 If it is desired to write Reports on a tape unit other than Logical Tape Unit Six, insert the desired tape number in this field. If the field is left blank, Reports will be written on Logical Tape Unit Six. The tape number may be changed by modifying the System Attribute OTAPE by source language statements.

49 through 54 The number of lines per page of printed output is inserted in this field. The maximum number of lines is 60. If the field is left blank, the number of lines will be 55. This may be changed at any time by changing the value of the System Attribute LINES.

The format for the System Specification Card is included at the top of the Initialization Form (see Fig. 24, Chapter 14).

Initial Conditions Deck

The Initial Conditions Deck consists of all Initialization Cards and Data Cards, as described in Chapter 14. As mentioned above, the Initial Conditions Deck may be read from tape units other than that containing the object program by indicating the desired tape unit on the System Specifications Card.

Blank Card

The Initial Conditions Deck must be followed by a blank card even if the Exogenous Event Deck is to be read from a different tape.

Exogenous Events Deck

The preparation of the Exogenous Events Deck was described in the discussion of the READ statement in Chapter 7.

Chapter 13

DEFINITION CARDS

Each type of Temporary Entity, Event Notice, Permanent Entity, Attribute, Set, and written Function must be described on the Definition Form shown in Fig. 23. The Form is divided into four panels. The first is used for defining Temporary Entities, Event Notices, and Temporary Attributes; the second for defining Permanent Entities and Permanent Attributes; the third for defining Sets; and the fourth for specifying the name and mode of any written Functions.

Entries may appear in any or all panels of a particular Definition Card. Blank panels are ignored. If a Definition Card appears in the Definition Deck with a blank in Col. 72, a translator–provided subprogram package will be compiled for each definition on the card. If there is a mark in Col. 72, none of the packages will be compiled. All Definition Cards must be included in the Definition Deck, regardless of whether or not their translator–provided packages have been previously compiled.

DEFINITION OF TEMPORARY ENTITIES

For each type of Temporary Entity, a "T" is inserted in Col. 2, and its source language name in Cols. 4 through 8 starting in Col. 4. The name may consist of one to five letters or numbers (one of which must be a letter). It may not end in "F," or contain "XX" or a special character such as $, +, etc.

Columns 9 through 17 specify the size of the memory record required for storing the Attribute values describing the particular type of Temporary Entity. This record consists of one "Master Record" plus, if desired, one to eight "Satellite Records." Master Records and Satellite Records may be one, two, four, or eight words long, the length being specified by writing "1," "2," "4," or "8" in the appropriate column of the Definition Form. If a Satellite Record is not needed, its record–size is left blank.

The Master Record must contain at least as many words as there are Satellite Records. For each Satellite Record specified, the translator uses the second half of the corresponding storage word in the Master Record; it may not be used for storing Attribute values. For example, if Satellite Records 1, 2, and 3 are specified, the Master Record must be at least four words long (a three–word record is not permitted), and the second half of the first, second, and third words of the Master Record may not be used when assigning locations to Attribute values. Particular numbered Satellite Records may be skipped so long as the preceding rules are followed.

The use of Satellite Records should be avoided when possible, since their storage and retrieval times are greater than those of Master Records. (These times are discussed later in the present Chapter.)

DEFINITION OF EVENT NOTICES

The Event Notice is a special kind of Temporary Entity, provided to facilitate the scheduling of Endogenous Events. Its only difference from a Temporary Entity is that the first two words of its Master Record are used by the CAUSE and CANCEL statements and the Timing Routine; therefore these two words may not be used for storing Attribute values. The number of words allocated to the Master Record of an Event Notice must always be at least two (and the first two words must not be used). Satellite Records 1 and 2 are unavailable to Event Notices.

The definition of an Event Notice is distinguished from that of a Temporary Entity by the insertion of an "N" in Col. 2 of the Definition Form. Its name and record size are specified according to the rules previously described for Temporary Entities, except that the first two words of the Master Record are already assigned, and the first two Satellite Records may not be used.

DEFINITION OF TEMPORARY ATTRIBUTES

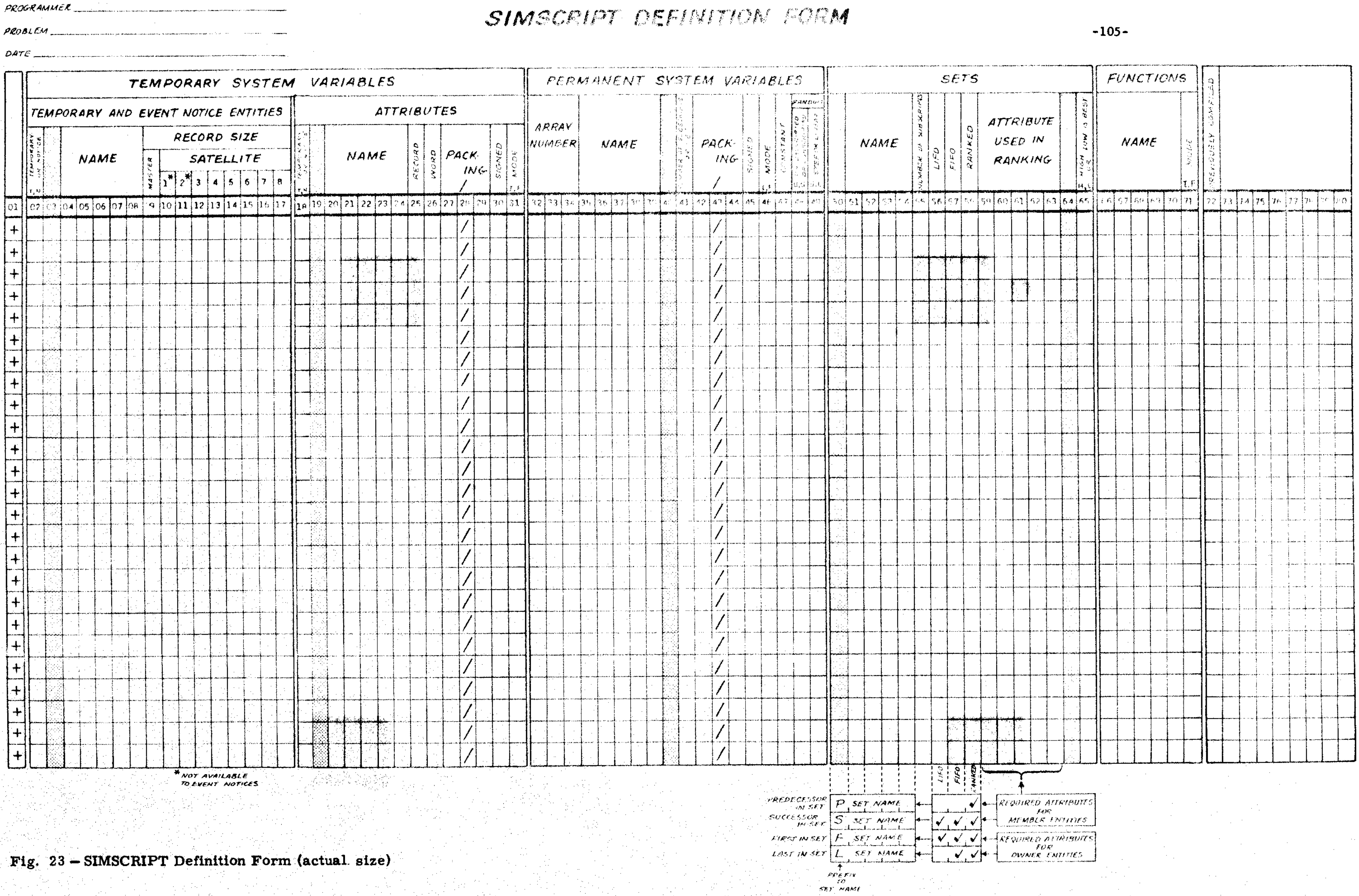
Each Attribute of a Temporary Entity or Event Notice is defined in Cols. 18 through 31 of the Definition Form. A Temporary Attribute definition may precede or follow the definition of its Entity, or it may appear on the same card.

Whether an Attribute describes a Temporary Entity or an Event Notice is indicated by inserting a "T" or an "N" respectively in Col. 18.

The Attribute name is inserted starting in Col. 20. It may consist of one to five letters or numbers, one of which must be a letter. It may not end in "F" or contain "XX" or a special character.

Each Attribute is stored in a fixed position within its Entity's record. This position is specified in the columns headed "Record," "Word," and "Packing."

Under "Record," one indicates in which record the Attribute is to be stored by inserting a zero (or blank) if the Attribute is to be stored in the Master Record, or by inserting the number of the Satellite (1, 2, . . . 8) if it is to be stored in a Satellite Record.



PROGRAMMER ______________________

PROBLEM ______________________

DATE ______________________

SIMSCRIPT DEFINITION FORM

TEMPORARY SYSTEM VARIABLES

TEMPORARY AND EVENT NOTICE ENTITIES: TEMPORARY OR NOTICE T,N · NAME · RECORD SIZE: MASTER · SATELLITE 1* 2* 3 4 5 6 7 8

ATTRIBUTES: TEMPORARY OR EVENT T,N · NAME · RECORD · WORD · PACKING / · SIGNED · MODE I,F

PERMANENT SYSTEM VARIABLES: ARRAY NUMBER · NAME · NUMBER OF SUBSCRIPTS · PACKING / · SIGNED · MODE I,F · CONSTANT · RANDOM

SETS: NAME · NUMBER OF SUBSCRIPTS · LIFO · FIFO · RANKED · ATTRIBUTE USED IN RANKING · HIGH OR LOW IS BEST H,L

FUNCTIONS: NAME · MODE I,F

PREVIOUSLY COMPILED

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

*NOT AVAILABLE TO EVENT NOTICES

		LIFO	FIFO	RANKED	
PREDECESSOR IN SET	P SET NAME			✓	REQUIRED ATTRIBUTES FOR MEMBER ENTITIES
SUCCESSOR IN SET	S SET NAME	✓	✓	✓	
FIRST IN SET	F SET NAME	✓	✓	✓	REQUIRED ATTRIBUTES FOR OWNER ENTITIES
LAST IN SET	L SET NAME		✓	✓	

PREFIX TO SET NAME

Fig. 23 — SIMSCRIPT Definition Form (actual size)

In the column headed "Word," one indicates which word within the record is to be used in storing the Attribute (inserting "1" if it is the first word, "2" if it is the second, etc.).

If it is desired to store more than one Attribute per storage word, the Attribute's position within the word is indicated in the columns headed "Packing." The permissible fractional word storage allocations and their corresponding codes are listed in Table 2. Any non–overlapping combination of these fractional spaces may be used. If overlapping storage assignments are made, there is no protection. However, if during a run of the object program an Attribute value exceeds the storage space allotted to it, the program will exit, and the computer will print an error message.

Table 2

MEMORY–PACKING CODES FOR TEMPORARY ATTRIBUTES AND SINGLE–SUBSCRIPTED PERMANENT ATTRIBUTES

Packing Code	Portion of Word where Attribute will be Stored	Maximum Integer Value		Approximate Limits for Non-zero Floating Point Value	
		Unsigned	Signed	Unsigned	Signed
Blank or 1/1	full word	131,071	131,071	2^{-128} to 2^{127}	2^{-128} to 2^{127}
1/2	first half	131,071	131,071	2^{-32} to 2^{31}	2^{-16} to 2^{15}
2/2	second half	131,071	131,071	2^{-32} to 2^{31}	2^{-16} to 2^{15}
1/3	first third	4,095	2,047	NOT PERMITTED	
2/3	second third	4,095	2,047		
3/3	last third	4,095	2,047		
1/4	first quarter	511	255		
2/4	second quarter	511	255		
3/4	third quarter	511	255		
4/4	fourth quarter	511	255		

A special restriction on storage allocations is that the first bit of the first word of a Master or Satellite Record must remain empty throughout the execution of the program; the remainder of the word may be used so long as the values being stored will not require use of the first bit.

This restriction prohibits storing a signed number in the first part of the first word (or in the whole word). It also limits the permissible values for unsigned numbers to those specified for signed numbers in Table 2. This restriction applies only to the first word of a Record, and then only to the case of a full word or if the packing code is 1/2, 1/3, 1/4. A convenient way to deal with this particular nuisance is to always use the first word for storing Attributes whose values are Entity identification numbers (for example, Attributes describing Set membership or ownership).

Identification numbers of Temporary Entities and Event Notices (being equal to actual memory locations) always require at least one-half word of storage. Identification numbers of Permanent Entities, being ordinal positions in a list, may be stored in a third or fourth of a word, depending on how many Entities there are of the particular type. Floating point numbers also require at least a half-word of storage. Alpha-numeric information must always be stored in a full word.

The packing procedure for Master and Satellite Records is the same except that when Satellite Records are used, only the first half of the corresponding numbered word in the Master Record is available for storing Attribute values.

Temporary Attribute storage and retrieval times for the various fractional word storage assignments are summarized in Table 3. These access times are expressed in machine cycles and are currently applicable to the IBM 709 and 7090 versions of SIMSCRIPT. As can be seen, packing should not be used if there is ample space for full-word storage, since it increases storage and retrieval times. Furthermore, storage and retrieval times are greater if the Attribute value is stored in a Satellite Record than if it is stored in a Master Record.

If the Attribute is to be "signed," a mark is placed in Col. 30; otherwise, the translator will assume it is always positive.

Indicate the mode of the Attribute value in Col. 31 by inserting "I" for integer or "F" for floating point.

ARRAY NUMBERS

As described above, Temporary Attributes are stored in individual memory records, each record containing all the Attributes of a particular Entity. Permanent Attributes, on the other hand, are stored as zero-, one-, or two-dimensional arrays, each array consisting of a single kind of Attribute.

The position of a particular Permanent Attribute within an array corresponds to the ordinal position of the particular Permanent Entity it describes. For example, if ITEM is a kind of Permanent Entity, the various

Table 3

STORAGE AND RETRIEVAL TIMES FOR TEMPORARY ATTRIBUTES

Packing Code	Stored in Master Record				Stored in Satellite Record			
	Integer		Floating Point		Integer		Floating Point	
	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed
Blank or 1/1	13 / 13	13 / 13	13 / 13	13 / 13	18 / 18	18 / 18	18 / 18	18 / 18
1/2	13 / 16	13 / 16	30 / 21	30 / 22	18 / 21	18 / 21	35 / 26	35 / 27
2/2	20 / 16	20 / 16	33 / 21	33 / 22	27 / 21	27 / 21	40 / 26	40 / 27
1/3	23 / 18	26 / 19	▨	▨	28 / 23	31 / 24	▨	▨
2/3	23 / 18	26 / 21	▨	▨	28 / 23	31 / 26	▨	▨
3/3	24 / 19	27 / 21	▨	▨	29 / 24	32 / 26	▨	▨
1/4	23 / 18	26 / 19	▨	▨	28 / 23	31 / 24	▨	▨
2/4	21 / 16	24 / 21	▨	▨	26 / 21	29 / 26	▨	▨
3/4	23 / 18	26 / 22	▨	▨	28 / 23	31 / 27	▨	▨
4/4	24 / 19	27 / 21	▨	▨	29 / 24	32 / 26	▨	▨

NOTE:
Computer cycles to store → 23 / 17 ← Computer cycles to retrieve

Attributes of the fifth ITEM would consist of the fifth entries in a number of different one-dimensional arrays.

An Attribute of a pair of Permanent Entities, such as transit times from one depot to another, would be stored as a two-dimensional array. Ragged Tables are also stored as two-dimensional arrays.

An unsubscripted or System Attribute is, in effect, a zero-dimensional array.

Each type of Permanent Attribute must be assigned an "Array Number" in Cols. 32 through 34 of the Definition Form (the units position must be in Col. 34). This Array Number is also used to identify each type of

Permanent Attribute on the Initialization Cards discussed in Chapter 14. The Array Number never appears in the source program.

If two or more types of Permanent Attributes are to be packed in the same word of storage, they must be given the same Array Number. As discussed later in the present Chapter, "equivalent" Permanent Attributes must also have the same Array Number. Otherwise, the Array Number must be unique for each type of Permanent Attribute.

Array Numbers need not be sequential on the Definition Form; however, numbers that are skipped must be included in the Initial Conditions deck just as though they had been assigned. They also use at least one word of storage each during execution of the object program.

DEFINITION OF PERMANENT ENTITIES

For each kind of Permanent Entity, an "E" is inserted in Col. 41. The name of the Entity type is inserted in Cols. 35 through 39; it may consist of one to five letters or numbers, one of which must be a letter. It may not end in "F" or contain "XX" or a special character. As discussed in Chapter 11, the translator automatically defines an unsubscripted Permanent Attribute with the name "NEntity name."

An Array Number for the System Attribute "NEntity name" is assigned in Cols. 32 through 34 of the line used to define the type of Permanent Entity. In setting the initial conditions for object program execution, this Array Number identifies the Attribute "NEntity name" on the Initialization Card.

The Array Number assigned to "NEntity name" must have a lower value than the Array Numbers assigned to any Attributes of the Permanent Entity.

DEFINITION OF PERMANENT ATTRIBUTES

The Array Number for each kind of Permanent Attribute is assigned in Cols. 32 through 34.

The name of the Attribute is inserted in Cols. 35 through 39. The name may consist of one to five letters or numbers, one of which must be a letter. It may not end in "F" or contain "XX" or a special character.

To indicate the number of subscripts the Attribute is to have, "0," "1," or "2" is inserted in Col. 41. A blank in Col. 41 is treated as a zero.

Unsubscripted Permanent Attributes (System Attributes) are always stored in a full word, i. e., packing is not allowed. Single-subscripted Permanent Attributes may be packed into the same word provided that they describe the same Permanent Entity. If the Attribute's value is to be stored in a fractional word, the desired packing is indicated in Cols. 42 and 44. If packing is not required, these columns may be left blank. Packing codes for single–subscripted Permanent Attributes and the permissible Attribute values were previously shown in Table 2.

If a particular kind of Permanent Attribute is assigned to the packing code, of say, "1/2," each successive value in the Attribute Array will be stored in the first half of each of a sequence of storage words. The number of words equals the number of Entities of the type that the Attribute describes. It also equals the value of the translator–defined System Attribute, "NE–ntity name." In the above example, a different type of Permanent Attribute could be stored in the second half of each word by giving it the same Array number and assigning the packing code "2/2." Any of the other fractional word assignments may also be made so long as they do not overlap.

Packing for double–subscripted Permanent Attributes is specified by placing a "2" or "4" in Col. 44, indicating that each entry is to be assigned a half or fourth of a word respectively (one–third packing is not permitted). Entries from successive columns of the two–dimensional Attribute Array will then be packed into the same word. Different types of double–subscripted Permanent Attributes are never packed together; therefore eacn has a unique Array Number and Col. 42 must be left blank.

The storage and retrieval of unsubscripted Permanent Attribute values each require six computer cycles. Storage and retrieval times for single– and double–subscripted Permanent Attributes are shown in Table 4.

If a Permanent Attribute is to be signed, a mark is placed in Col. 45; otherwise it is always assumed to be non–negative.

In Col. 46, the mode of the Attribute is indicated by inserting an "I" for integer or an "F" for floating point.

If the initial values of a particular type of Permanent Attribute are not to be changed during execution of the object program, the generated subprogram for storing the values can be deleted from the object program by inserting a mark in Col. 47.

DEFINITION OF RANDOM ATTRIBUTES

Permanent Attributes whose values are to be automatically obtained by one of the random look–up table procedures described in Chapter 11

Table 4

STORAGE AND RETRIEVAL TIMES FOR SUBSCRIPTED PERMANENT ATTRIBUTES

Packing Code	Single-Subscripted				Double-Subscripted			
	Integer		Floating Point		Integer		Floating Point	
	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed	Unsigned	Signed
Blank or 1/1	14 / 14	14 / 14	14 / 14	14 / 14	20 / 20	20 / 20	20 / 20	20 / 20
1/2	14 / 17	14 / 17	31[b] / 22[c]	31[b] / 23[c]	26 / 32	26 / 32	44[d] / 39[e]	44[d] / 40[e]
2/2	23 / 17	23 / 17	36[b] / 22[c]	36[b] / 23[c]	33 / 32	33 / 32	46[d] / 39[e]	46[d] / 39[e]
1/3	25 / 19	28[a] / 20						
2/3	25 / 19	28[a] / 22						
3/3	26 / 20	29[a] / 22						
1/4	25 / 19	28[a] / 20			42 / 36	45[a] / 36		
2/4	23 / 17	26[a] / 22			41 / 35	44[a] / 37		
3/4	25 / 19	28[a] / 23			42 / 36	45[a] / 38		
4/4	26 / 20	29[a] / 22			43 / 37	46[a] / 38		

NOTE:
Computer cycles to store → 25 / 14 ← Computer cycles to retrieve

[a] Storage requires 2 cycles less than indicated if value is non-negative.

[b] Storage requires 12 cycles less than indicated if value is zero.

[c] Retrieval requires 4 cycles less than indicated if value is zero.

[d] Storage requires 6 cycles less than indicated if value is zero.

[e] Retrieval requires 5 cycles less than indicated if value is zero.

are defined in the manner described above, except as specified below. Instead of specifying the number of subscripts in Col. 41, indicate whether the Attribute is to be subscripted or unsubscripted by inserting "S" or "U" in Col. 48. If there is an entry in Col. 48, Col. 41 must be blank. Double-subscripted Attributes are not permitted with random look-up table procedures.

In Col. 49, indicate whether the step function or the linear interpolation procedure is desired by inserting "S" or "L."

If a random look–up procedure is not desired, Cols. 48 and 49 are left blank.

EQUIVALENCE

As might be inferred from the preceding discussion, the same Temporary or Permanent Attribute may be assigned more than one name.

With Temporary Attributes, "equivalence" is achieved by assigning the same "Record," "Word," and "Packing" codes to the equivalent Attributes. With Permanent Attributes, "equivalence" is achieved by assigning the same Array Number and Packing code to equivalent Attributes.

DEFINITION OF SETS

Each kind of Set must be defined in the third panel of the Definition Form.

The name of the Set is inserted in Cols. 51 through 54 starting in Col. 51. It may consist of any one to four letters or numbers except that it may not end in "F" or contain "XX" or a special character. The Set name is limited to four characters because it must be given a one–letter prefix to form the names of certain required Attributes. These Temporary or Permanent Attributes must in turn be defined for the types of Entities owning and belonging to the Set.

Column 55 indicates the number of subscripts the Set is to have (0, 1, or 2).

The type of Set organization is indicated in Col. 56, 57, or 58. If the member Entities are to be inserted and removed on a last–in–first–out, first–in–first–out, or ranked basis, a mark is inserted under LIFO, FIFO, or Ranked, respectively.

If the Set is a Ranked Set, the name of the Attribute according to whose value the member Entities are to be ranked is inserted in Cols. 59 through 63 starting in Col. 59. This Temporary or Permanent Attribute must also appear among the Attribute definitions for the member type of Entity. Any Attribute may be used for this purpose. If the Set is ranked, it is further indicated in Col. 65 whether a high or a low value of the "priority" Attribute is considered best (insert "H" for high, "L" for low).

A reminder of the special Attributes that must be defined for the member and owner types of Entity has been printed at the bottom of the Definition Form. The Attribute names that are formed by prefixing a letter to the Set name are enumerated. The check marks below the columns marked

LIFO, FIFO, and Ranked indicate which of these Attributes are required for each type of Set organization.

DEFINITION OF FUNCTION NAMES

For each written Function subprogram, the name and mode of the Function must be defined in the last panel of the Definition Form. The names are formed according to the same rules as those for naming System Variables; the mode is indicated by inserting "I" for integer or "F" for floating point.

Chapter 14

INITIAL CONDITIONS DECK

The specification of initial conditions at the start of a simulation could vary in complexity from the input of a complete status description at one extreme to the mere specification of the initial values of Permanent Attributes at the other extreme. The Initial Conditions Deck does the latter. *

The Initialization Deck consists of Data Cards plus Initialization Cards punched from the Initialization Form shown in Fig. 24. Every Array Number from "1" up to the largest appearing in Cols. 32–34 of the Definitions Deck must be accounted for in sequential order in the Initialization Cards. The complete sequence of Array numbers must be accounted for, even though some may have been omitted from the Definitions Deck or may have appeared out of sequence.

Procedures for preparing the Initial Conditions Deck are discussed under the following headings:

Unsubscripted Permanent Attributes
Single–Subscripted Permanent Attributes
Double–Subscripted Permanent Attributes
Ragged Tables
Random Look–Up Tables for Unsubscripted Attributes
Random Look–Up Tables for Subscripted Attributes

UNSUBSCRIPTED PERMANENT ATTRIBUTES

Each unsubscripted Permanent Attribute (System Attribute) defined on the Definition Form must have its initial value read in or set equal to zero. It is also necessary to read in values for each of the translator–defined Attributes with the name "NEntity name", thereby establishing the number of each type of Permanent Entity. The procedure for specifying the values of the translator–defined Attributes "NAttribute name (I)" is described in this Chapter under the discussion of Ragged Tables. Other

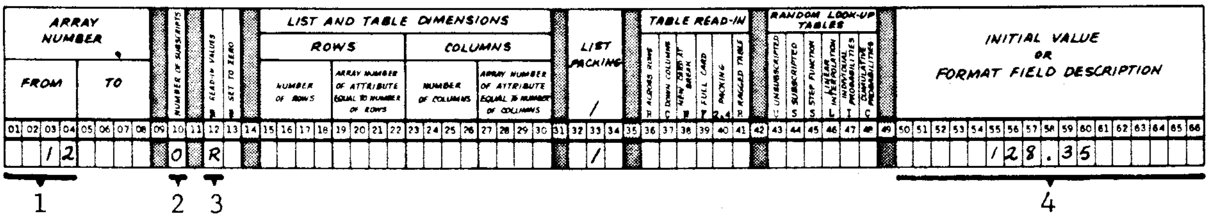
* If a more complete initial status description is desired, a representative mix of each type of Temporary Entity and Event Notice in various stages of their life spans may be specified and values assigned to their Attributes. The Set relationship among the various Entities in existence at the given point in time may also be specified. Such a snapshot of a complete system Status might be obtained either from an analysis of empirical data or from the results of a previous simulation run. A special Exogenous Event routine can be written for the purpose of setting these various conditions at the start of the simulation.

translator—defined Attributes described in Chapter 11, such as TIME, PAGE, RANDM, etc., do not require initial values although some of them may be specified by means of the System Specification Card.

Initial values of unsubscripted Permanent Attributes may be separately specified by means of individual Initialization Cards. They may also be handled in groups by means of a single Initialization Card followed by Data Cards. To be initialized as a group, the System Attributes in the group must have consecutive Array Numbers. Their values must also be read in by using the same FORMAT statement Field Description.

Figure 25 shows the entries required to read in the initial value of a single System Attribute.

The initial value can be set to zero by inserting a zero in Cols. 50 through 66, or by leaving Col. 12 blank and inserting a "Z" in Col. 13. A Format Field Description and Data Card can also be used to specify the initial value as described below for the case of more than one System Attribute.



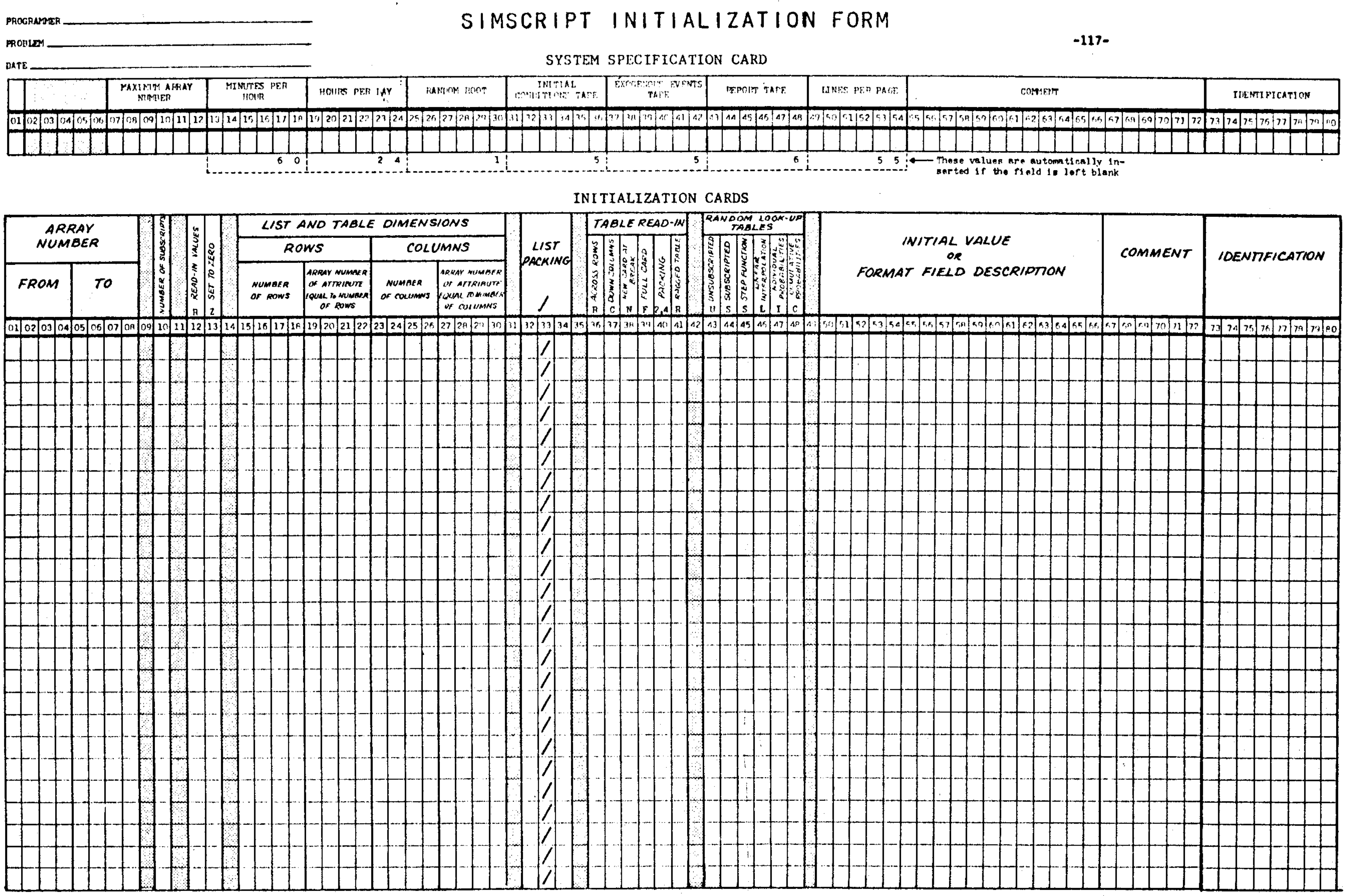
1 - Enter Array Number in Cols. 1 through 4. The units position of the Array Number must be in Col. 4.

2 - Enter a zero in Col. 10.

3 - Enter an "R" in Col. 12.

4 - Enter the Initial Value as an integer or decimal number anywhere in Cols. 50 through 66. Formats other than integer or decimal (e.g., hours or alphanumeric) must be read from Data Cards as shown in Fig. 26.

Fig. 25 — Initialization Card Entries for a Single Unsubscripted Permanent Attribute



PROGRAMMER ____________________

PROBLEM ____________________

DATE ____________________

SIMSCRIPT INITIALIZATION FORM

SYSTEM SPECIFICATION CARD

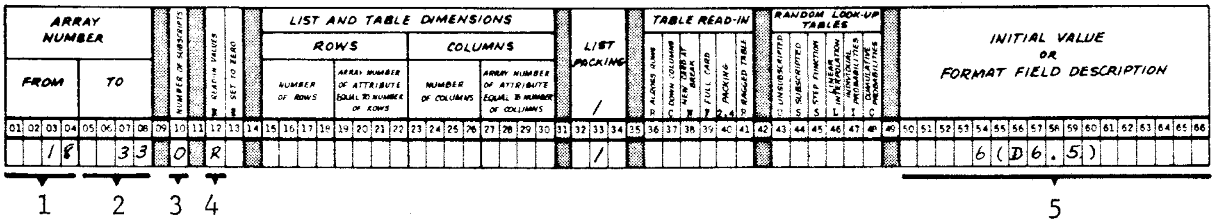
		MAXIMUM ARRAY NUMBER	MINUTES PER HOUR	HOURS PER DAY	RANDOM ROOT	INITIAL CONDITIONS TAPE	EXOGENOUS EVENTS TAPE	REPORT TAPE	LINES PER PAGE	COMMENT	IDENTIFICATION
01	02 03 04 05 06	07 08 09 10 11 12	13 14 15 16 17 18	19 20 21 22 23 24	25 26 27 28 29 30	31 32 33 34 35 36	37 38 39 40 41 42	43 44 45 46 47 48	49 50 51 52 53 54	55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72	73 74 75 76 77 78 79 80
			6 0	2 4	1	5	5	6	5 5	← These values are automatically inserted if the field is left blank	

INITIALIZATION CARDS

ARRAY NUMBER								LIST AND TABLE DIMENSIONS								TABLE READ-IN							RANDOM LOOK-UP TABLES								
								ROWS		COLUMNS																					
FROM	TO		NUMBER OF SUBSCRIPTS		READ-IN VALUES	SET TO ZERO		NUMBER OF ROWS	ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF ROWS	NUMBER OF COLUMNS	ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF COLUMNS		LIST PACKING		ACROSS ROWS	DOWN COLUMNS	NEW CARD AT BREAK	FULL CARD	PACKING	RAGGED TABLE		UNSUBSCRIPTED	SUBSCRIPTED	STEP FUNCTION	LINEAR INTERPOLATION	INDIVIDUAL PROBABILITIES	CUMULATIVE PROBABILITIES		INITIAL VALUE OR FORMAT FIELD DESCRIPTION	COMMENT	IDENTIFICATION
					R	Z							/		R	C	N	F	2,4	R		U	S	S	L	I	C				
01 02 03 04	05 06 07 08	09	10	11	12	13	14	15 16 17 18	19 20 21 22	23 24 25 26	27 28 29 30	31	32 33 34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66	67 68 69 70 71 72	73 74 75 76 77 78 79 80
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		
													/																		

Fig. 24 — SIMSCRIPT Initialization Form (actual size)

The Initialization Card entries required for reading in the initial values of a series of unsubscripted Permanent Attributes with consecutive Array Numbers are shown in Fig. 26.


ARRAY NUMBER
FROM
TO
NUMBER OF SUBSCRIPTS
READ-IN VALUES R
SET TO ZERO Z
LIST AND TABLE DIMENSIONS
ROWS
NUMBER OF ROWS
ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF ROWS
COLUMNS
NUMBER OF COLUMNS
ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF COLUMNS
LIST PACKING
/
TABLE READ-IN
ACROSS ROWS R
DOWN COLUMNS C
NEW CARD AT BREAK N
FULL CARD F
PACKING 2,4
RAGGED TABLE R
RANDOM LOOK-UP TABLES
UNSUBSCRIPTED C
SUBSCRIPTED S
STEP FUNCTION S
LINEAR INTERPOLATION L
INDIVIDUAL PROBABILITIES I
CUMULATIVE PROBABILITIES C
INITIAL VALUE OR FORMAT FIELD DESCRIPTION
01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
1
2
3
4
5

18 | 33 | 0 | R | / | 6(D6.5)

1 - Enter the smallest Array Number in Cols. 1 through 4.

2 - Enter the highest Array Number in Cols. 5 through 8.

3 - Enter a zero in Col. 10 (or leave blank).

4 - Enter an "R" in Col. 12.

5 - Enter a single FORMAT statement Field Description enclosed in parentheses in Cols. 50 through 66. The initial values will be read according to this Field Description from Data Cards following the Initialization Card.

Fig. 26 — Initialization Card Entries for a Series of Unsubscripted Permanent Attributes

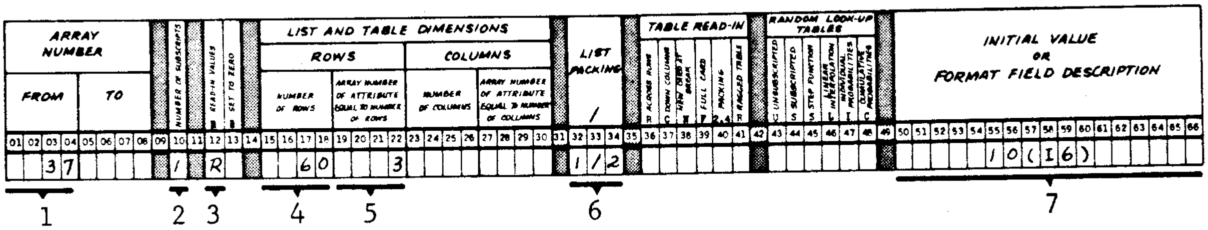
Although only one Field Description is permitted, the field may be repeated across the Data Card by prefixing the optional constant to the field Description in the manner described in Chapter 7. The first field of each Data Card starts in Col. 1, and data may appear in all columns up through Col. 72.

If all the values of a group of consecutively numbered System Attributes are to be set initially to zero, the lowest and highest Array Numbers are indicated in Cols. 1 through 8, and a "Z" is inserted in Col. 13. The remainder of the Initialization Card is left blank and Data Cards are not required.

SINGLE-SUBSCRIPTED PERMANENT ATTRIBUTES

If the initial values are to be read in, a separate Initialization Card followed by Data Cards is required for each list of single-subscripted Permanent Attributes. If the initial values are to be set equal to zero, one or more lists of single–subscripted Permanent Attributes may be handled by a single Initialization Card, provided the lists are of the same length and have consecutive Array Numbers.

To read in the initial values of a list of single–subscripted Permanent Attributes, the Initialization Card entries shown in Fig. 27 are required.



1 - Enter the Array Number in Cols. 1 through 4

2 - Enter a "1" in Col. 10.

3 - Enter an "R" in Col. 12.

4 - In Cols. 15 through 18, enter the largest value the subscript is to take on.

5 - In Cols. 19 through 22, enter the Array Number of the System Attribute "NEntity name" corresponding to the type of Entity that the Attribute list describes. The value of "NEntity name" must have been previously read in from an Initialization Card or Data Card. This value must be the same as that of the largest subscript as specified in Cols. 15 through 18. These two values are automatically compared as a consistency check.

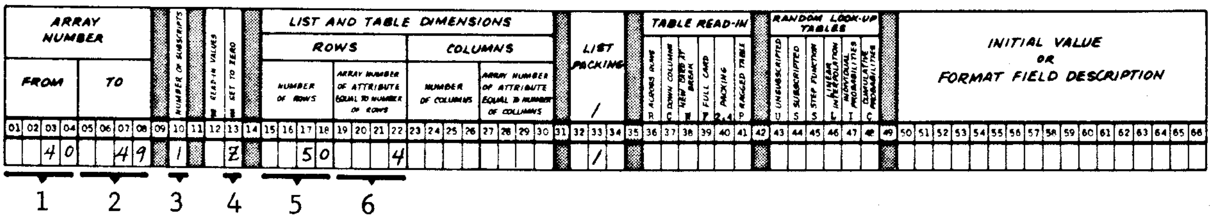
6 - If fractional word packing was called for on the Definition Form, enter this same packing code in Cols. 32 and 34.

7 - In Cols. 50 through 66, enter a single FORMAT statement Field Description enclosed in parentheses and preceded by an optional constant, if desired. This Field Description tells how the initial values of the list are to appear in the subsequent Data Cards. Each Data Card will be read starting in Col. 1. If desired, successive values may appear across the Data Card as described in the discussion of Field Descriptions in Chapter 7.

Fig. 27 – Initialization Card Entries for Reading In a Single–subscripted Permanent Attribute List

One or more lists of single–subscripted Permanent Attributes describing the same Entity and having consecutive Array Numbers can be initially set equal to zero by the Initialization Card entries shown in Fig. 28.

Inserting the letter "Z" in Col. 13 causes zeros to be stored in the entire word, irrespective of any packing that may have been called for on the Definition Form. To initialize packed, floating point variables see the discussion at the end of the Chapter.



1 - Enter the lowest Array Number in Cols. 1 through 4.

2 - Enter the highest Array Number in Cols. 5 through 8.
(If there is only one list, leave columns blank).

3 - Enter a "1" in Col. 10.

4 - Enter a "Z" in Col. 13.

5 - Indicate largest subscript value in Cols. 15 through 18.

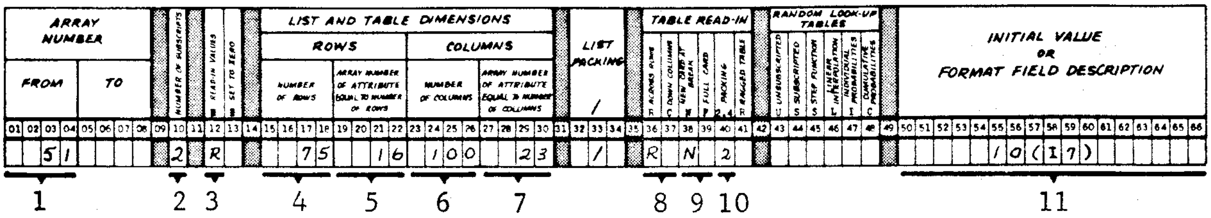
6 - Enter the Array Number of the appropriate System Attribute "NEntity name" in Cols. 19 through 22.

Fig. 28 – Initialization Card Entries for Setting Single–subscripted Permanent Attribute Lists to Zero

DOUBLE–SUBSCRIPTED PERMANENT ATTRIBUTES

If non–zero initial values are to be read in for a table of double–subscripted Permanent Attributes, each table requires a separate Initialization Card followed by Data Cards containing the values. However, a single Initialization Card may serve to zero out one or more Attribute tables, provided they all describe the same pair of Permanent Entities and have consecutive Array Numbers. The procedure for setting Ragged Tables equal to zero is described below.

Figure 29 shows the Initialization Card entries required for reading in the initial values of a table of double–subscripted Permanent Attributes.



ARRAY NUMBER		NUMBER OF SUBSCRIPTS	READ-IN VALUES	SET TO ZERO	LIST AND TABLE DIMENSIONS				LIST PACKING	TABLE READ-IN						RANDOM LOOK-UP TABLES						INITIAL VALUE OR FORMAT FIELD DESCRIPTION
					ROWS		COLUMNS															
FROM	TO				NUMBER OF ROWS	ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF ROWS	NUMBER OF COLUMNS	ARRAY NUMBER OF ATTRIBUTE EQUAL TO NUMBER OF COLUMNS	/	R ACROSS ROWS	C DOWN COLUMNS	N NEW CARD AT BREAK	F FULL CARD	2,4 PACKING	R RAGGED TABLE	U UNSUBSCRIPTED	S SUBSCRIPTED	S STEP FUNCTION	L LINEAR INTERPOLATION	I INDIVIDUAL PROBABILITIES	C CUMULATIVE PROBABILITIES	
01 02 03 04	05 06 07 08	10	12	13	15 16 17 18	19 20 21 22	23 24 25 26	27 28 29 30	32 33 34	36	37	38	39	40	41	43	44	45	46	47	48	50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66
51		2	R		75	16	100	23	/	R		N		2								10(I7)
1		2	3		4	5	6	7		8		9		10								11

1 - Enter the Array Number in Cols. 1 through 4.

2 - Enter a "2" in Col. 10.

3 - Enter an "R" in Col. 12.

4 - In Cols. 15 through 18, indicate the largest value the row subscript will take on. (The first subscript of a double-subscripted Permanent Attribute always designates the row of the table.)

5 - In Cols. 19 through 22, enter the Array Number of the System Variable "NEntity name", the value of which is equal to the value of the largest row subscript.

6 - In Cols. 23 through 26, indicate the largest column subscript.

7 - In Cols. 27 through 30, enter the Array Number of the System Variable "NEntity name", the value of which is equal to the value of the largest column subscript.

8 - Indicate the order in which the Attribute values are to be read from Data Cards by entering an "R" in Col. 36, if the values are to be read across rows, or entering a "C" in Col. 37 if they are to be read down columns.

9 - If the beginning of each new row or column is to start on a new Data Card, enter "N" in Col. 38. If, instead of starting on a new card, the first entry in a new row or column immediately follows the last entry in the preceding row or column, put an "F" in Col. 39.

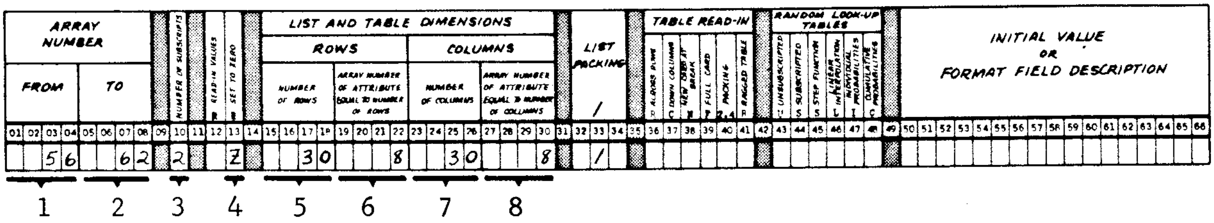
10 - If the Table entries are to be packed into a half or a fourth of a storage word, inset "2" or "4" respectively in Col. 40. Leave this column blank if packing is not desired. The packing designation must agree with that indicated on the Definition Form. (Columns 32-34 of the Initialization Card are ignored in the case of double-subscripted Permanent Attributes.)

11 - In Cols. 50 through 66, enter a FORMAT statement Field Description enclosed in parentheses indicating how the table entries are to appear in subsequent Data Cards.

Fig. 29 — Initialization Card Entries for Reading In a Double—subscripted Permanent Attribute Table

Figure 30 shows the initialization entries required to zero out one or more tables of Permanent Attributes describing the same pair of Permanent Entities and having consecutive Array Numbers.

To initialize packed, floating point variables, see the discussion at the end of the chapter.



1 - Enter the lowest Array Number in Cols. 1 through 4.

2 - Enter the highest Array Number in Cols. 5 through 8. (If there is only one table, leave columns blank.)

3 - Enter a "2" in Col. 10.

4 - Enter a "Z" in Col. 13.

5 - Enter the number of rows in Cols. 15 through 18.

6 - Enter the Array Number of "NEntity name" for the first dimension in Cols. 19 through 22.

7 - Indicate the number of columns in Cols. 23 through 26.

8 - Enter the Array Number of the corresponding System Attribute "NEntity name" in Cols. 27 through 30.

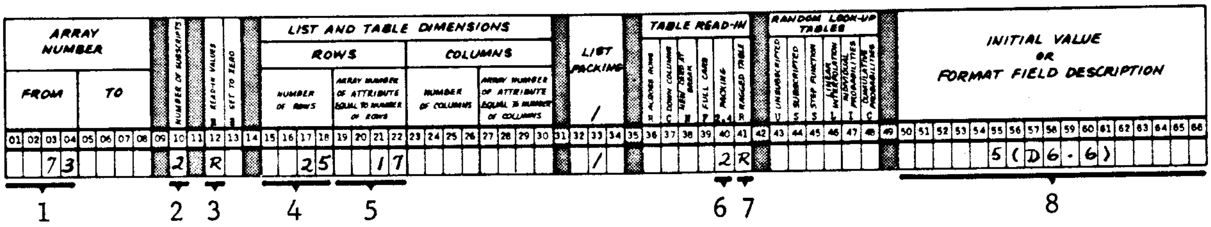
Fig. 30 – Initialization Card Entries for Setting Double–subscripted Permanent Attribute Tables to Zero

RAGGED TABLES

Permanent Entities may be described by Attributes having more than one value, in which case the Attributes are treated as a two–dimensional "Ragged Table" with rows of variable length. Each Ragged Table requires one Initialization Card followed by Data Cards containing its initial values. If a Ragged Table is to be initially set to zero, zeros must be read in from Data Cards.

Figure 31 shows the Initialization Card entries required to read in a Ragged Table.

Since row–length varies, Cols. 23 through 30 of the Initialization Card are left blank, and the length of each row is indicated in the Data Cards. The letter "C" is punched in Col. 72 of the Data Card if the row is continued on the following card. In the Data Card containing the end of the row, the number of fields to be read is punched in Cols. 71 and 72. In reading Ragged Tables, Cols. 71 and 72 of the Data Cards may never be used for data. Values are always read across rows, and each row starts on a new card. To initialize packed floating point variables see the discussion at the end of the Chapter.



1 - Enter the Array Number in Cols. 1 through 4.

2 - Enter "2" in Col. 10.

3 - Enter "R" in Col. 12.

4 - Enter the number of rows in Cols. 15 through 18.

5 - Enter the Array Number of the corresponding System Attribute "NEntity name" in Cols. 19 through 22.

6 - If a half-word or quarter-word packing is desired, put "2" or "4" respectively in Col. 40.

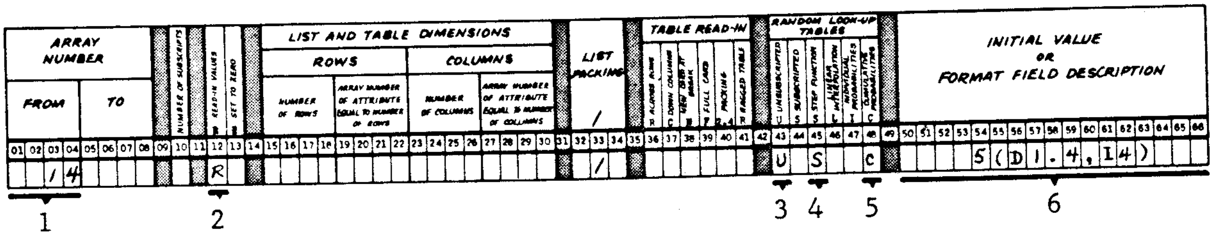
7 - Enter "R" in Col. 41.

8 - Enter a Field Description in Cols. 50 through 66, specifying how the table values are to appear in the Data Cards.

Fig. 31 – Initialization Card Entries for Reading In a Ragged Table

RANDOM LOOK–UP TABLES FOR UNSUBSCRIPTED ATTRIBUTES

Permanent Attributes whose values are to be determined by one of the random look–up procedures described in Chapter 11 do not have initial values, of course; however, the look–up tables from which their values will be determined must be input as part of the Initial Conditions Deck. Each look–up table requires one Initialization Card followed by Data Cards describing the particular table. Figure 32 shows the Initialization Card entries required to input a look–up table for an unsubscripted Permanent Attribute.



1 - Enter the Array Number of the Attribute in Cols. 1 through 4.

2 - Enter "R" in Col. 12.

3 - Enter "U" in Col. 43.

4 - If the Random Attribute values are to be generated by the step function procedure, put "S" in Col. 45; if by linear interpolation, put "L" in Col. 46.

5 - Indicate whether the look-up table is expressed in terms of individual or cumulative probabilities by placing "I" in Col. 47 or "C" in Col. 48, respectively. (Only cumulative probabilities may be used for the linear interpolation procedure.)

6 - Enter a pair of FORMAT statement Field Descriptions enclosed in parentheses in Cols. 50 through 66.

Fig. 32 – Initialization Card Entries for a Random Look–up Table for an Unsubscripted Permanent Attribute

The first of the two Field Descriptions in Cols. 50 through 66 must describe a decimal field; it indicates how the individual or cumulative probabilities are to appear in the subsequent Data Cards. The second Field Description describes how the values corresponding to the probabilities are to appear in the Data Cards. In the case of the step function procedure, the value may be of any mode other than alpha–numeric. In the case of the linear interpolation procedure, the value must be floating point (i. e., decimal, decimal hours, or days, hours, and minutes). See the discussion of packed, floating point variables at the end of the chapter.

As described under the discussion of FORMAT statement in Chapter 7, these pairs of entries maybe repeated across a Data Card by prefixing to the Field Descriptions the number of cases (pairs of entries) per card. Pairs of entries may not be split at the end of a Data Card.

The end of the look–up table is indicated in the Data Cards in the same manner as that described for Ragged Tables. The letter "C" is punched in Col. 72 of all but the last Data Card. The number of pairs of entries to be read from the last Data Card is punched into Cols. 71 and 72 of that card; these columns may not be used for any other purpose.

Cumulative probabilities must appear in the Data Cards in order of increasing cumulative value. If individual probabilities are read in, they will be automatically accumulated in the order in which they appear. In either case, the final cumulative probability is automatically set equal to 1.0 irrespective of the probability values in the Data Cards. In the linear interpolation procedure, the input of individual probabilities is not permitted, and the value of the first cumulative probability must be 0.0.

RANDOM LOOK–UP TABLES FOR SINGLE–SUBSCRIPTED ATTRIBUTES

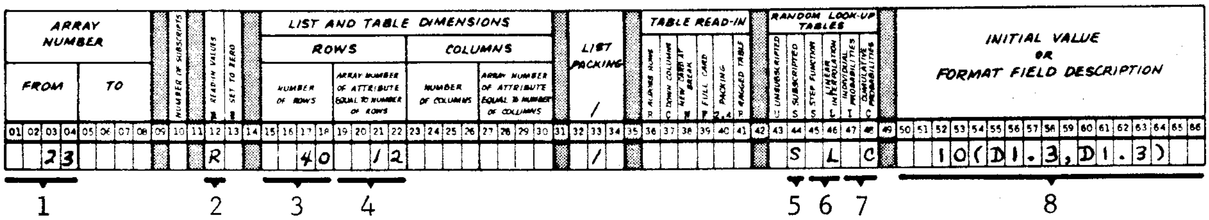
If a random look–up procedure is to be used to determine the values of a single–subscripted Permanent Attribute, a separate look–up table is required for each value the subscript can take on. One Initialization Card followed by Data Cards containing this subscripted series of look–up tables must be included in the Initialization Deck. The required Initialization Card entries are shown in Fig. 33.

Each look–up table in the subscripted series must begin on a new Data Card. The end of each table is indicated by punching the number of paired entries to be read from the last Data Card in Cols. 71 and 72 of that card. The letter "C" is punched in Col. 72 of the other Data Cards. Columns 71 and 72 may not be used for any other purpose.

INITIALIZATION OF PACKED, FLOATING POINT VARIABLES

The binary representation of a decimal number packed into half of a word depends on whether or not provision is made for negative values. For example, the number 1.0 is represented by "010 001 100 000 000 000" if the half–word represents a "signed" floating point variable, and by "100 001 100 000 000 000" if the half–word represents an "unsigned" floating point variable.

If "half" packing and a "D" FORMAT statement field description are specified on the Initialization Form, the initialization routine assumes that signed floating point variables are to be read in and packed. In the



1 - Enter the Array Number in Cols. 1 through 4.

2 - Enter "R" in Col. 12.

3 - Enter the maximum subscript value in Cols. 15 through 18.

4 - Indicate the Array Number of the System Attribute "NEntity name" whose value is equal to the maximum subscript value.

5 - Enter "S" in Col. 44.

6 - Enter either "S" in Col. 45 for "step function", or "L" in Col. 46 for "linear interpolation".

7 - Enter "I" in Col. 47 for "individual probabilities" or "C" in Col. 48 for "cumulative probabilities".

8 - Enter a pair of Field Descriptions enclosed in parentheses in Cols. 50 through 66. (Two separate Field Descriptions must be specified. In the above example, 10(2D1.3) is not permitted.)

Fig. 33 — Initialization Card Entries for a Random Look—up Table for a Single—subscripted Permanent Attribute

case of "unsigned", packed floating point variables, a "U" must be used in place of the "D"

The specifications on the Initialization Form must agree with those of the Definition Form (in other words, if Col. 45 is not marked on the Definition Form, the U format must be used on the Initialization Form and conversely). The U format is used only on the Initialization Form and is not required or permitted in FORMAT statements in the source program. Decimal—hours and days, hours, and minutes field descriptions may be used on the Initialization Form just as they are used in FORMAT statements since time values must be positive and are stored as unsigned floating point numbers.

In specifying the two fields of a random look—up table, the first field (which always contains the probability) may be described by either a D or U field description (D and U are equivalent here). The second field containing the value of the random variable must be described by a D field description if the value is to be a signed floating point number; it must be described by a U field description if the value is to be an unsigned floating point number.

Chapter 15

MEMORY LAYOUT

While it is not essential for the programmer to know how memory is arranged in SIMSCRIPT object programs, an explanation of memory layout may be of interest in itself, may help clarify certain restrictions in the source language, or may be of use to the experienced programmer intending some unconventional application. *

The object program generated by SIMSCRIPT divides memory into four main regions, as illustrated in Fig. 34.

Region I, the Program Region, contains the various FORTRAN object subprograms generated by the SIMSCRIPT translator; the lower part of Region I is used by the FORTRAN system. The various kinds of subprograms described in Chapter 9 and any specified FORTRAN Library Routine are stored in the upper part of Region I, above the "FORTRAN Break." Local Variables are stored in this region in their respective subprograms.

Skipping Regions II and III for the moment, consider Region IV located at word 31,000 to the end of memory at 32,767. The upper part of Region IV is reserved for the standard use of FORTRAN. The lower part of Region IV from the beginning of "COMMON," at 32,561, down to 31,000 is used for various data needed by the object programs but not defined by the SIMSCRIPT programmer. It includes two input buffers for the Exogenous Event Tape, the current value of TIME, the times and types of the most imminent Exogenous and Endogenous Events, and the like. The only information in this region which the SIMSCRIPT programmer may refer to are certain pre–defined variables, such as TIME, LINES, PAGE, etc.

Region III is used to store all Permanent Attribute Arrays and Random Look–Up Tables, including:

Zero–dimensional arrays: individual numbers, referred to in the source program by unsubscripted Permanent Attributes (sometimes called System Attributes).

One–dimensional arrays: lists of numbers referred to in the source program as single–subscripted Permanent Attributes.

* In debugging, it should rarely ever be necessary to dump memory. Instead, interim results should be progressively checked by special "diagnostic" Reports which are easily obtained by means of the Report Generator

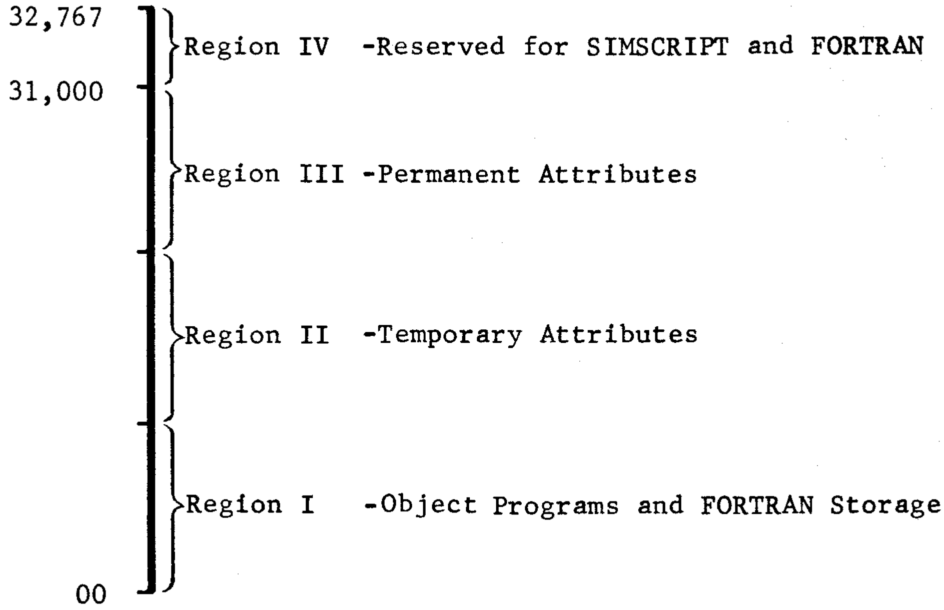

32,767
31,000
00
Region IV -Reserved for SIMSCRIPT and FORTRAN
Region III -Permanent Attributes
Region II -Temporary Attributes
Region I -Object Programs and FORTRAN Storage

Fig. 34 – SIMSCRIPT Memory Layout

Two–dimensional arrays: tables of numbers referred to as double–subscripted Permanent Attributes. These may be regular tables in which all rows have the same length, or "ragged" tables with rows of different length.

Random Look–Up Tables: probability distributions for randomly determining the values of unsubscripted or single–subscripted Permanent Attributes as described in Chapter 11.

The amount of space allocated to one– or two–dimensional arrays and to Random Look–Up Tables is not specified when the object program is compiled. Instead, it is specified as part of the initial conditions at the time the object program is executed. Thus, an array may contain five entries for one run and 100 entries for a second run without loss of space or need to recompile in either case. This facility requires that the memory layout scheme for Region III allow a particular list to start in different places depending on the length of the preceding lists.

Suppose, for concreteness of discussion, that there are 37 arrays, i. e., that 37 is the maximum Array Number appearing on the Definition Form. Memory word number 31000–1 (i. e., word 30999) would be used on behalf of Array Number 1, word 31000–2 on behalf of Array Number 2, and so on through word 31000–37, used on behalf of Array Number 37. If the first array is zero–dimensional — in other words, a single number — then this number is stored at 30999. If the fifth array is a one—dimensional list, and word 30995 (i. e., 31000—5) should happen to contain the number 29,000 in the address portion of the word, this would indicate that the first element of the fifth array is at 28999, the second element at 28998, the third at 28997, and so on. *

In general, if C(y) represents the contents of word y, then (C(31000—L)—I) is where the Ith element of the Lth list is to be found.

If the sixth array happens to be a two—dimensional table, then C(31000—6) is once again the location immediately above the array. The first element of the table, however, is not at C(31000—6) — 1; rather, C(30994)—1 contains the zero—th location of the first row of the table, C(30994)—2 contains the zero—th location of the second row, and so on. ** This organization is used for both regular and "ragged" tables.

In the case of a Random Look—Up Table for an unsubscripted Permanent Attribute with Array Number "A", C(31000—A) contains the location immediately above the location of the start of the distribution. The distribution itself is represented by storing cumulative probability and value of outcome in the manner described in Chapter 11 and 14.

Random Look—Up Tables for subscripted Permanent Attributes are stored in a manner somewhat akin to Ragged Tables with C(31000—A)—I equal to the location immediately above the first work of Ith distribution.

The various data references described above are accomplished automatically on the basis of the contents of the Initialization Cards and need not concern the programmer.

In simulation programs, three additional one—dimensional arrays used by the Timing Routine are also stored at the lower end of Region III.

Consider next the organization of Region II, which is used for storing the Attributes of Temporary Entities and Event Notices. All available space from the end of the last array at the bottom of Region III to the end of the

* The number indicating the zero-th location of a one- or two-dimensional array is always preceded by a tag of "2". In the above example, the contents of word 30995 would consist of the octal representation of 29000 preceded by the number "2".

** These locations are stored in the decrement, and the length of the row is stored in the address portion of the word. There are no tags.

last subprogram at the top of Region I is used for storing Temporary Attributes.* The number of each type of Temporary Entity in existence changes during the course of program execution. The memory arrangement in Region II must therefore be capable of allocating space when a Temporary Entity or Event Notice is created during the course of the simulation. It must also be able to reclaim such space for subsequent use when a Temporary Entity or Event Notice is destroyed.

The values of Temporary Attributes are held in "records". An Entity may have as many as nine records for its Attributes, and each record may consist of either one, two, four, or eight consecutive words of core storage. Of the one or more records which characterize an Entity, one record is referred to as the Master Record, and the other records, if any, are referred to as Satellites.

Every Entity of a particular type has the same configuration of records. Thus if a FLITE is a Temporary Entity with an eight-word Master Record and one four-word Satellite, every FLITE created during program execution is assigned an eight-word Master and a four-word Satellite. Two FLITEs in existence at the same time will of course be given different words of memory for their records.

The identification number of a Temporary Entity is equal to the zero-th location of its Master Record, e.g., the statement CREATE DOG CALLED FIDO finds available space to form the records required to characterize a DOG as specified on the Definition Form. It also sets the variable FIDO equal to the identification number of the DOG just created. The location of the first word of the Master Record is at FIDO-1, the second word is at FIDO-2 and so on. It is not necessary for the programmer to refer to FIDO-2 since each Attribute's position within the Record is specified on the Definition Form. If an Attribute called BDAY (for Birthday) is assigned in the first half of the second word of the Master Record describing a DOG, the mention of BDAY(FIDO) in any subprogram will result in the computer retrieving or storing the desired value from the second half of the word located at FIDO-2.

Satellite Records will not necessarily be adjacent to their Master Records. The location of the Ith Satellite is to be found in the second half of the Ith word of the Master Record. More precisely, if there is an Ith Satellite then the second half of the Ith word of the Master Record contains the zero-th location of the Satellite Record. Consequently, if there is an Ith

[*] After the Initialization Cards have been read in, that portion of the System Package which accomplishes the Initialization is set to zero and this space is included as part of Region II.

Satellite, the second half of the Ith word of the Master Record is not available for storing Attribute values. The linkages between the Master Record and its Satellites are automatically set up whenever a CREATE statement is executed.

It is not necessary for Satellite Records to be consecutive. In other words, it is permissible to have a "Satellite 8" for a particular type of Entity even though it has no Satellites 1 through 7. It is mandatory, of course, that there be an eight–word Master Record if there is a "Satellite 8." In general, if there is a Satellite "I" the Master Record must contain at least "I" words.

The identification number based on the location of the Master Record is always used as a subscript even when referring to information stored in a Satellite record. Thus, the source program would refer to SIZE(FIDO), whether the Attribute SIZE is stored in the Master Record or in a Satellite Record of the DOG called FIDO.

Since each Master and Satellite Record must be 1, 2, 4, or 8 words long, it is necessary for the CREATE routines to have access to unused memory sections of 1, 2, 4, and 8 consecutive words in length. The method by which the SIMSCRIPT object program keeps track of available–ones, available–twos, available–fours, and available–eights is described next.

First consider the available–eights, pretending for the moment that records of other sizes are never called for. At the beginning of object program execution (after Region III has been set up and its initial values read in) Region II is, in effect, broken into available–eights. A variable in Region IV is set equal to the location immediately below the location of the first available eight. The first word in this available–eight record is set equal to the location of a second available–eight. The first word of the second record similarly contains the location of a third available–eight, etc., and the first word of the last available–eight contains a zero rather than a location, thus indicating the end of the set available–eights. When an eight is requested during the course of a run, the first available–eight (as indicated by the pointer stored in Region IV) is made available to the requesting routine. This "first–available–eight–pointer" is then set equal to the next available–eight, as indicated by the first word of the previous first–available–eight. When an eight–word record is returned to available storage, the reverse occurs. The record just returned becomes first, while the original first is noted in its first word as its successor available–eight.

Suppose now that a four–word record is requested. The program will check a memory location in Region IV which contains the location immediately below the first available–four. The first time an available–four is requested, the "four–pointer" will equal zero, indicating that no four–word records are available. In this case, the first "eight" will be removed from the available–eights and split into two "fours." One of these will be given to the routine as a "four", and the other will be put into the set of available–fours. In general, whenever a "four" is requested it is supplied from the set of available–fours. If the set is empty, an "eight" is split in two. Whenever a "four" is returned, the program determines whether its "brother–four" (the "other four" that was made out of the same "eight" as the one being returned) is in the set of available–fours. If it is, the two "fours" are combined into an "eight" and returned to the set of available–eights.

Similarly, "twos" are made from "fours" when needed, and returned to "fours" whenever a pair of "brother–twos" become available. Available–ones in turn are made from and returned to available–twos.

This dividing and recombining of records is facilitated by two conventions. First, in originally forming the set of available–eights a few words of memory are wasted, if necessary, so that every "eight" can have a zero–th location whose binary representation ends in "111." Consequently, the binary representation of "fours" ends in "111" or 011." The "brother" of any "four" may be determined by complementing the third bit from the right. The "eight" from which a "four" was originally obtained and whence it goes, may be determined by setting this third bit equal to one. A similar situation holds for "twos" and "ones", with the second and first bits, respectively, playing the crucial roles.

A second convention facilitating the recombining of records is that the first bit of the first word of an available "one," "two," or "four" is set equal to one if the record is not in use. Thus, the DESTROY routines can quickly find, by checking a bit, whether the brother of a record is available or not. As a consequence, however, this bit may never be used for data (see the discussion of this restriction in Chapter 13).

To facilitate the removal of an available "one," "two," or "four" from the middle of its respective sets of available records when its brother also becomes free, the <u>preceding</u> available record in the set, as well as the <u>succeeding</u> available record, is noted in the decrement and address portions of the first word of each available record.