Oxford University Computing Laboratory

Computer Manuals

ICL 1900 Control and Simulation Language SIMON

4138

CONTROL & SIMULATION

Simulation Language - SIMON

1900

ICL

International Computers Limited
©

OXFORD UNIVERSITY COMPUTING SERVICE

User notice

MANUAL (NOTICE NO.) 11/6/6[illegible]

4138 SIMON (1)

OXFORD UNIVERSITY COMPUTING LABORATORY
Copy 1 COMPUTING SERVICE 4138

File one copy of this notice with each of the manuals indicated.

CONTENTS

ALSIMONB

SIMON is a simulation language which consists of a set of procedures written in ALGOL. It is provided source form on punched cards or on paper tape.

A new version of SIMON is now available and will be issued to all holders of the package. It is known as ALSIMONB to distinguish from the earlier version which is now known as ALSIMONA. ALSIMONB corrects errors that were discovered in ALSIMONA, several of the procedures have been speeded up and new procedures have been added. At the same time a second edition of the SIMON manual (Technical Publication 4138) has been produced.

Copies of ALSIMONB may be obtained from

Mrs B. Phillips
Issue Team
Software Issue Branch
International Computers Limited
30/31 Friar Street
READING RG1 1JP
Berks

When ordering please state whether the paper tape version or the card version is required, or both. The routine GENER, for generating pseudo-random numbers, requires bit manipulation. This routine is written in PLAN and is available as a paper tape catering for 15-bit address mode and 22-bit address mode;

FORM 1/230/45(3.69)

[illegible]d by Technical Publications Service, 30 - 31 Friar Street, Reading, Berkshire. RG1 1JP

alternatively, it is available on cards in 15-bit address mode in semi-compiled form. Again, when ordering, please state the version or versions required.

Copies of the new manual may be obtained from

Literature Distribution Branch
International Computers Limited
30/31 Friar Street
READING RG1 1JP
Berks

The operating instructions for ALSIMONA and ALSIMONB differ and it is emphasised that the old manual refers to ALSIMONA, and the new one to ALSIMONB.

Distribution of the new version is expected to begin shortly, and users not receiving it within 14 days of the publication date of this notice should inform Mrs Phillips at the address above.

CORRECTIONS TO SIMON MANUAL (SECOND EDITION)

The following errors should be corrected in the second edition of the SIMON manual (Technical Publication 4138).

Page 9: line -1 should read

```
gwritehist(table(8('table')'));
```

Page 28: under inposition(n,list), the words "defined by list" should be deleted from the Remarks column.

© International Computers Limited, Reading,1969

ICL International Computers Limited

User notice

Library

RECEIVED 2 2 MAY 1972

MANUAL (NOTICE NO.)

23/5/72

4138 SIMON (2)

File one copy of this notice with each of the manuals indicated.

ERRORS IN MANUAL

Certain errors have been discovered in the SIMON manual; these are listed below.

Page 9

The specification of procedure *ghist* should read:

ghist (name,lowerbound,n,width)

The final line of this page and the amended version given in User Notice 1 are both incorrect. The inner parentheses of the original version should be deleted; alternatively, the 1900 Series hardware representation is as follows:

```
GWRITEHIST(TABLE,'('TABLE')');
```

Page 11

The formula given on page 11 for calculation of the size of master lists is incorrect as it does not allow for the fact that entities which are members of group entities require three places on the list. The smallest allowable list size is

$$2e + 3f + s + m$$

where s and m are as before, e is the number of simple entities and f the number of entities within group entities.

Example

On the outline flowchart, the box on page 16 which reads "customer joins queue" should read "customer leaves timeset and joins queue". The box on page 17 which starts "Haircut begins..." should include "Behead queue." On page 18, *delete (member) from:(timeset);* should follow the line starting "*phase a*".

FORM 1/230/4813.69

Issued by Technical Publications Service, 30 - 31 Friar Street, Reading, Berkshire. RG1 1JP

The following errors on page 18 should also be noted:

1 (third line of program) Assignment of values 1 to 10 to *zzxzz* will not give satisfactory random numbers (see page 23).

2 The ninth line of the program should read:

```
integer timeset,leasttime,member,clocktime,r,s,t
```

3 In the final seven lines of the page, the Algol basic symbols if, then, else, begin and end should have been printed in bold typeface.

4 The following line should appear before the penultimate line of the page.

```
addlast (customer[0],timeset);
```

On page 19, the lines:

```
settime (r,t+clocktime);
settime (r,s+clocktime);
```

should appear in the reverse order. Note also that there are more efficient ways in which this piece of the program could have been coded.

Page 25

The sixth line following the heading *Sequence of input* should read:

```
for i:=1 step 1 until 10 do zzxzz[i]:=8000000-i*50;
```

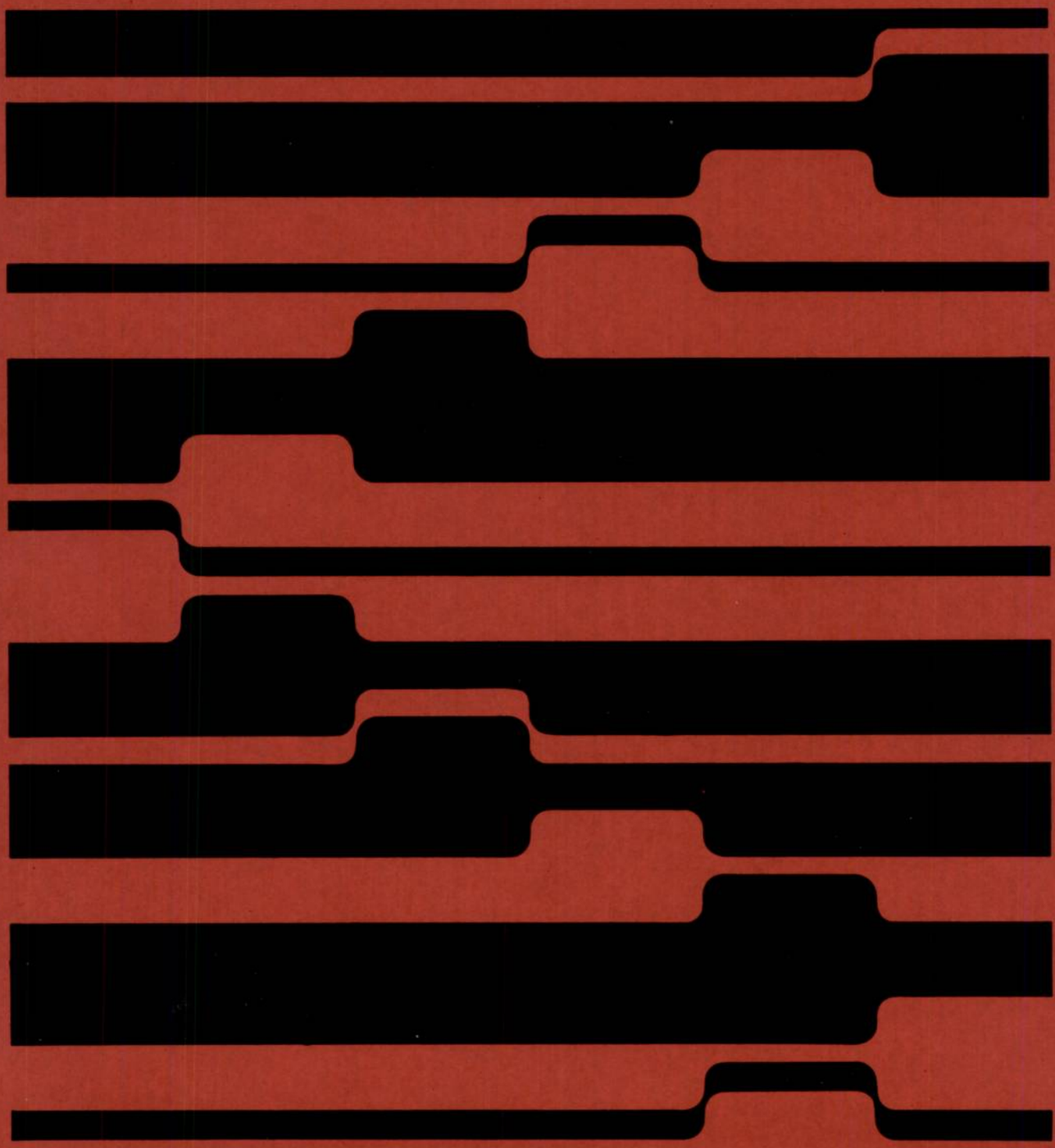
© International Computers Limited, Reading, 1972

ICL

Simulation Language SIMON

1900 Series

OXFORD UNIVERSITY COMPUTING LIBRARY
Copy 1.
COMPUTING SERVICE 4138



ICL

Simulation Language SIMON

1900 Series

OXFORD UNIVERSITY COMPUTING LABORATORY
Copy 1. COMPUTING SERVICE 4138

The policy of International Computers Limited is one of continuous development and improvement of its products and services, and the right is therefore reserved to alter the information contained in this document without notice. ICL makes every endeavour to ensure the accuracy of the contents of this document but does not accept liability for any error or omission. Any equipment or software performance figures and times stated herein are those which ICL expects to be achieved in normal circumstances. Wherever practicable, ICL is willing to verify upon request the accuracy of any specific matter contained in this document.

Technical Publication 4138

©International Computers Limited 1966

First Edition March 1966

Second Edition January 1969.

Issued by Technical Publications Service
International Computers Limited
Head Office: ICL House, Putney, London SW15
Produced by ICL Printing Services
at Letchworth, Hertfordshire

Preface

This manual describes SIMON, a simulation language in Algol developed by P.R. Hills at Bristol College of Science and Technology. Simulation is a technique for obtaining information about the performance of a system by constructing a model which behaves in the same way as the system in all essential details. The technique is usually applied where items passing through a system form queues at certain stages while waiting to be processed. In general, the sizes of these queues and the time taken for items to pass through the system vary in a random way. The basic requirement for a model of such a system is a means of keeping ordered lists of items, and updating these lists in chronological order. The SIMON language provides special facilities for this purpose.

SIMON is available to any user of the ICL 1900 Series of computers who has Algol compilers. This manual does not attempt to be an introduction to the art of simulation and model building, but merely gives enough information about SIMON procedures to enable an experienced simulator to use them. It is also assumed throughout the manual that the reader has a working knowledge of Algol and can refer to an Algol manual for specific details.

The manual is arranged as follows:

The Introduction gives an outline of the uses of simulation. Chapter 1 describes the type of model formulated in a SIMON program, and the method used for setting up and running the model. The basic terms used in the manual are explained.

Chapter 2 gives a full list of SIMON procedures with details of the use of each.

Chapter 3 describes the preparation of data for SIMON programs and the form of the results which may be obtained.

Chapter 4 gives a simple example program, with the data provided and results obtained.

Chapter 5 describes the implementation of SIMON for the 1900 Series.

Chapter 6 describes the preparation of a simulation run using ALSIMONB, the current 1900 pack of SIMON procedures.

This manual supersedes the first edition of SIMON: *I.C.T. Control and Simulation Manual Part I SIMON*, published in March 1966. The following publications by the originator of SIMON should be consulted for further information on SIMON:

P.R. Hills *"Simon, a computer simulation language in Algol."* Bristol College of Science and Technology, 1964.

P.R. Hills *"Simulation Language in Algol."* Digital simulation in O.R. - Proceeding of a Conference, Hamburg 6-10 September 1965. Edited by S.H. Hollingdale.

ICL wishes to acknowledge and thank P.R. Hills who originated the SIMON language, and who, by providing programs, test data and documentation, gave valuable assistance in the development and testing of SIMON procedures for the 1900 Series computers. ICL has taken the liberty of altering P.R. Hills' coding for certain procedures.

OXFORD U[illegible]RY
Copy 1. [illegible] 4130

Contents

Introduction

Simulation is a technique for obtaining information about the performance of a system without actually putting that system into operation. A model is constructed so that the results obtained by operating or modifying the model indicate the results to be expected when the corresponding system is operated or modified.

The technique has a wide range of application in operational research and can be used to study such varied activities as industrial processes, accountancy systems or marketing methods. In particular, the effect of many different approaches or strategies can be evaluated without implementing new systems or modifying those in existence.

Simulation is seldom applied to stationary systems. It is of most interest and value when it is applied to systems that are affected by the passage of time, and in almost all such systems some sort of queueing takes place. A queue is formed when a number of items have to wait before passing through some stage in a process. The rate at which the queue grows depends on the rate at which items join the queue, and the time taken to process each item. Both these factors may vary in a random, or stochastic, way; the problem is to derive information about the way in which the queue size varies. For example, the average length of the queue may be obtained, or the probability of its exceeding a certain length. If there is any degree of complexity in the process - if, for example, there are a number of inter-related queues - a mathematical analysis becomes impossibly complicated.

The simulation technique is to create a model of the system by making lists of items at each stage in the process, and transferring items from one list to another in the correct chronological order. All the information required about the process can then be obtained by observation of the model. This technique can be carried out by hand; however, for any but the most simple cases it becomes so cumbersome that the use of a computer is essential.

Languages such as FORTRAN and Algol have only rudimentary listing facilities, and consequently a number of special languages such as SIMON and C.S.L. have been developed. In addition to simplifying the writing of simulation programs, these languages may well find many applications outside the field of simulation, because of the powerful facilities they provide for examining and rearranging the membership of lists.

In SIMON, the various operations used in simulation programs are written in Algol, and declared as Algol procedures. The procedures are written in a block of program which is supplied to the user on a reel of paper tape, or a deck of punched cards, and is read into the computer with the simulation program. The normal structure and syntax of Algol are used, and anyone familiar with Algol should find no difficulty in learning and using SIMON. An additional advantage of SIMON is that users may easily supplement or amend the standard set of procedures.

The model formulated by a SIMON program consists of a number of individual components, called entities, each of which may exist in a number of discrete states. These entities are chosen so that by describing the state of each entity at a given time, the state of the model at that time is defined in as much detail as required.

The states of entities are recorded in two ways: each entity may have numerical values associated with it, corresponding to various states; or lists may be kept of all the entities in particular states. The method used depends on how information is to be referenced. For example, suppose we have four entities A, B, C, D, each capable of taking any or all of the four states w, x, y, z. In order to choose a particular entity and determine the states associated with it, we could prepare a table as follows:

ENTITIES	A	B	C	D
	w	x	w	x
STATES	x	y	z	y
		z		

If, on the other hand, we wish to choose a particular state and determine which entities are in that state, we could prepare a table as follows:

STATES	w	x	y	z
	A	A	B	B
ENTITIES	C	B	D	C
		D		

Either or both of these methods may be useful in a simulation program.

Having set up a model, by defining the entities and states, and the lists which are to be made, a method is required for defining the way in which the model changes with the passing of time. At any time two classes of entities may exist: those which will change autonomously as a result of the passing of time and those which will change only as a result of changes in other entities. The former are referred to as time-dependent entities, and the latter as time-independent entities. Each time-dependent entity is assigned a time-value, which is the time at which it is due to change its state. Time is advanced through the simulation by moving forward to the earliest time at which a change occurs. At this time all the consequent changes in the model are carried out.

The interval of time between changes is frequently determined by a number of unpredictable factors, so that an accurate prediction of the interval is impossible. Instead, observations must be made to determine the probability of the time interval falling within various ranges of values. From such observations probability distributions are prepared, which are provided as data for the SIMON program. Time-values are then obtained by sampling from the appropriate distributions.

Because of the random variations which occur in the processes simulated, the results obtained from a simulation will usually have to be subjected to some kind of statistical analysis. To facilitate this, results may be collected and output as histograms, on the line printer or by any other medium.

For further information on the art of simulation, the following publications are recommended:

Thomas Naylor, Joseph Balintfy, Donald Burdick and Kong Chu	*"Computer Simulation Techniques."* John Wiley and Sons 1966
K.P. Tocher	*"The Art of Simulation."* D. Van Nostrand Co. 1963
Ed. H. Meyer	*"Symposium on Monte Carlo Methods."* John Wiley and Sons 1956
Philip Morse	*"Queues, Inventories and Maintenance."* John Wiley and Sons 1958

Chapter 1 Basic definitions

This chapter introduces and provides definitions of the basic concepts of SIMON. The Algol publication language is used throughout the manual, in accordance with ICL Algol manuals.

ENTITIES

The component parts of the model are called *entities*. The choice of entities is an important stage in defining a simulation problem. It is sometimes possible to group together several of the components of the real system into a single entity in the model. The degree to which the model can be simplified in this way will depend on how much detail is required in the results, and what factors are considered important enough to be taken into account in the model.

In SIMON, each entity is represented by a one-dimensional **integer array**.

The array is set up by a normal array declaration, and defined as an entity by the SIMON procedure: *entity*.

Each of the elements of the array represents some property or *attribute* of the entity, and the value assigned to the element indicates the state of that attribute at a particular time. Consider, for example, an entity representing a ship. Suppose two items of information are required about the ship: whether it is in port or at sea, and what tonnage it is. The entity could be declared

```
integer array ship [0:2];
```

and the second element of the array, *ship* [1], would take one of two values indicating that the ship is either in port or at sea. The third element, *ship* [2], would take a value representing the tonnage of the ship.

Although the states are stored as **integer** values in the array, these values may be assigned to variables and the variable names used to describe entity states in the program. Thus we could write

```
location := 1;
tonnage  := 2;
atsea    := 1;
inport   := 2;
```

and then the statements

```
ship [location] := at sea;
```

and

```
ship [1] := 1;
```

both have the effect of assigning to the second element of the array *ship* the value 1, indicating that the ship is at sea.

The element *ship* [0] is reserved for a value assigned automatically by the entity procedure. This element must be used to represent the entity when it is needed as a parameter to one of the SIMON procedures.

A number of entities with the same set of attributes may be defined collectively as a *groupentity*. A groupentity is represented by a two-dimensional **integer array**. This array is set up by a normal array declaration, and then defined as a groupentity by the SIMON procedure: *groupentity*.

Each row of the array represents a single entity, and each column represents one attribute common to all entities. Element 0 of each row is reserved for a value assigned by the *groupentity* procedure. This element must be used to represent the entity in the groupentity when it is needed as a parameter to one of the SIMON procedures. The other elements in each row can be used to hold the attributes of the entity it represents.

For example, a fleet of five ships could be represented by

```
integer array ships [1:5, 0:2];
```

and the information

ships [1, *tonnage*] : = 10000;

ships [1, *location*] : = *atsea;*

ships [3, *location*] : = *inport;*

would be stored as shown in the diagram below.

	Reserved Column 0		Location Column 1		Tonnage Column 2	
Ship 1	Values assigned by program	[1,0]	1	[1,1]	10000	[1,2]
Ship 2		[2,0]		[2,1]		[2,2]
Ship 3		[3,0]	2	[3,1]		[3,2]
Ship 4		[4,0]		[4,1]		[4,2]
Ship 5		[5,0]		[5,1]		[5,2]

SETS

Another way of storing the states of entities is to form lists of entities which are in particular states. These lists are called *sets.* For example a set called *atsea* could be established to contain the names of all ships currently at sea. Each state, as defined in this way, is necessarily two-valued, since an entity is either in the set or not in the set.

In order to record the change of state of an entity, the entity is added to or removed from the appropriate sets, without itself being altered.

If this method is used, the number of entities with a particular state may be found without examining the attributes of individual entities.

A set is formed, in SIMON, by declaring an integer variable, and then defining the variable name as the name of a set by means of the SIMON procedure: *set.* The name defined is then used as a parameter to all the procedures which operate on that set.

QUEUES

Most simulation problems are concerned with queues, that is collections of items waiting, in order, to be processed.

All sets in SIMON have the properties of queues. Items may be added to a set only in a fixed position, so that an order is established. Any item in a set at any time may be referenced by its position, and this facility enables queues to be simulated. An entity can appear as many times as required in the same set.

ACTIVITIES

The actions taken to change the state of the model are called *activities.* They fall into two classes referred to as *phase-B* and *phase-C* activities.

Phase-B activities consist of state-changes which take place as a direct result of the advance of simulation time. *Phase-C* activities consist of state-changes which take place as a result of other changes which have been made at that moment in simulation time. Thus *phase-C* activities are generally associated with tests, the results of which determine the changes that are required.

TIME ADVANCE

A simulation model exists in discrete states, so that only the times at which state-changes occur need be considered, and simulation time advances in discrete jumps.

In SIMON, time is advanced in the following way. Each time-dependent entity is assigned a time-value, which is the time at which the state of the entity will change. All time-dependent entities are listed in a timeset; this is not done automatically and the programmer must use one of the normal listing procedures. Before any activities can take place the timeset is scanned for the entity with the smallest time-value. The time-value of this entity is the earliest time at which any change in the model can take place, so that simulation time is advanced to that time. The appropriate activities can then be initiated.

The time-value to be assigned to an entity is obtained by adding to the current value of the simulation time, a time period which may be a *constant*, a *determinable variable*, or a *stochastic variable*.

Consider the case of unloading a ship. This process would be idealized by assuming that the cargo and the discharging operations do not vary in any way, so that the time to unload is a constant.

A more realistic approach would be to take into account, for example, the amount of cargo carried by the ship. Assuming a fixed relationship between the amount of cargo and the unloading time, the time would be determined by an arithmetic expression. This time would be a determinable variable.

In reality, the time taken to unload the ship would depend on various unpredictable factors, and would therefore be a stochastic variable. The time, in this case, would be obtained by sampling at random from a probability distribution of unloading times. Facilities which are provided in SIMON for obtaining values in this way are described in Chapter 2.

The time-values of entities which are not time-dependent can be used to hold the values of any other attribute which needs to be scanned.

SAMPLING

Probability distributions which are to be sampled are provided as data. Pairs of values are read, consisting of a time t, and a probability P, where P is the probability that an event will occur before the time t has elapsed. Thus, the values form a cumulative percentage probability distribution. In order to sample a time value from the distribution, a probability value is selected at random by means of a random number generator, and the corresponding time is found from the distribution, interpolating linearly where necessary.

There is, of course, nothing to prevent variables other than time being sampled in this way.

PROGRAM STRUCTURE

Although there is no restriction on program structure imposed by the language, most simulation programs fall naturally into three phases.

Phase-A consists of searching the timeset for the entity with the least time-value, and advancing the simulation time to the time found.

Phase-B consists of making such state-changes as are a direct result of the time-advance.

Phase-C consists of making any further changes that are required because of the new state of the model, set up by *phase-B*. As soon as changes are made in *phase-C*, further changes may be required, so that *phase-C* is normally repeated until the model reaches a new steady state when no further changes are possible until the time is advanced again.

Besides these three phases, the program must contain statements which set the model in the required initial state, and statements which collect and output data obtained in the simulation.

MONITORING

Input and output are carried out by the usual procedures though some additional facilities are provided in 1900 SIMON. SIMON also provides procedures for the collection of results in histograms, which may be printed out in tabular form when required. The same procedures automatically calculate and print out the mean, variance and standard deviation of the values in a histogram.

Chapter 2 The SIMON procedures

This chapter consists of a list of procedures which may be used in SIMON programs. The original SIMON procedures are written first. These are followed by additional procedures incorporated in ALSIMONA and ALSIMONB, which are extended versions of the language provided by ICL. See Chapter 5.

INITIATION

plantmaster

must be called before any of the procedures described below (except the distribution sampling and histogram procedures) may be used. This procedure sets up the chain lists (master lists) in which entities and sets are stored.

ENTITIES AND SETS

entity (name, reference number)

specifies that *name*, which must have been declared as a one-dimensional **integer array**, is to be used to represent an entity. *reference number* is an arbitrary **integer** chosen by the programmer. Its value may be referred to by the *refnum* procedure which is described on page 7.

The various listing procedures which operate on sets and entities require actual parameters with numerical values, so that *name* [0] rather than *name* is used as a parameter to such procedures. The **integer array** *name* must therefore always include the element [0], which must not be altered except by the SIMON procedures.

Examples:

```
integer array barber a [0:3];
entity (barber a, 3);

integer array balance [-4:4];
entity (balance, 7);
```

groupentity (name, number of members, reference number)

specifies that *name* is to be used to represent a group of similar entities. *name* must have been declared as a two-dimensional **integer array** *name* [1:*m*, 0:*n*]; where *m* is the number of members, and *n* is the number of attributes of each member. *number of members* gives the same value as *m*. *reference number* is an **integer** chosen by the programmer, and used in the same way as for *entity*. The **integer array** used for a groupentity must always have 1 to *m* rows, and a column 0. The elements in column 0 must not be altered except by the SIMON procedures.

Examples:

```
integer array ships [1:5, 0:3];
groupentity (ships, 5,2);

integer array balances [1:9, -4:4];
groupentity (balances. 9,6);
```

set (name)

specifies that *name*, which must have been delcared by an **integer** type declaration, is the name of a set. While *name* is being used to represent a set, its value must not be altered.

Example:

```
integer timeset, waiting line, empty ships;

set (timeset);
```

set (waiting line);

set (empty ships);

LISTING PROCEDURES

addfirst (name, setname)

adds *name* to the head of the specified set. *name* may represent an entity (in which case the first element of the array must be used in the call) or it may represent a set. A groupentity may not be added to a set: the members of the groupentity must be added individually.

Examples:

addfirst (s s pig [0], *empty ships);*

addfirst (ships [3,0], *empty ships);*

Here, as in other procedures, use is made of the Algol facility that the sequence:

) any string of letters : (

is equivalent to a comma in a parameter list.

For example:

addfirst (ships [3:0], *empty ships);*

could equally well be written

addfirst (ships [3:0] *) to: (empty ships);*

addlast (name) to: (setname)

adds *name* to the tail of the specified set.

behead (setname)

deletes the first member from the specified set. If the set is empty, a message is printed, and there is a jump to the label *stop.*

betail (setname)

deletes the last member from the specified set. If the set is empty, a message is printed and there is a jump to the label *stop.*

delete (name) from : (setname)

deletes the specified member from the specified set. *name* may represent an entity or a set. As usual, an entity must be represented by its first element; for example:

delete (s s dog [0] *) from : (empty ships);*

If *name* is not in the set, there is a paperthrow and a message is printed.

rotate (setname, number)

rotates the specified set, making the first member last, the second first, and so on, *n* times.

IMPLICIT NAMES

headof (setname)

tailof (setname)

refer to the first and last members of a set. These are Algol function procedures, and so may be used as parameters to other procedures. The result is an integer equal to the one in position 0 of the entity in the given position.

For example:

addfirst (tailof(set 1)) to: (set 2);

adds the last member of *set* 1 to the head of *set* 2, leaving *set* 1 unchanged. Implicit names may not be used to reference attributes.

For example:

tailof (set 1)[3] := atsea;

is illegal.

REFERENCE TO NUMERICAL VALUES

sizeof (setname)

is a function procedure which provides a value equal to the number of members in the specified set. This may be used in expressions, or as a parameter to other procedures. If the set is empty, a message is printed and there is a jump to the label *stop.*

For example:

if *sizeof (set 1) = 0* **then goto** *label3*
else *print (sizeof (set 1,4,0));*

memnum (name)

is a function procedure which provides the value of the first subscript for one of the members of a groupentity. The parameter *name* will usually be an implicit name. For example, if the first member of a set *empty ships* is the entity *ship* [5, 0], then the statement

x:=memnum(headof(emptyships));

would assign to *x* the value 5. This is a useful method of referring to an entity whose identity is unknown. For example the following statement could now be written:

ship [x, location] := atsea;

When applied to an entity, not a groupentity, *memnum* is equivalent to *refnum.*

refnum (name)

is a function procedure which provides the value of the reference number that the programmer has associated with the specified entity. *name* will usually be an implicit name. For example, if *member* refers to an entity with reference number 5, then

refnum(member)

provides the value 5.

This procedure has one particularly useful application; the *refnum* procedure may be used in place of an arithmetic expression in a switch designator. For example, suppose that a branch is to occur to *b1, b2, b3,* or *phase-C,* depending on which entity is selected by the *phase-A* scan. This could be carried out by the statements

switch *b:=b1, b2, b3, phasec;*
.....;...
goto *b [refnum (member)1];*

where each entity has the reference number 1, 2, 3, or 4, and *member* refers to the entity selected in *phase-A* of the program.

TIMEVALUES

settime(name) to: (value)

stores the given **integer** *value* as the time-value associated with the specified entity, *name. value* may be replaced by an expression giving an **integer** result.

For example:

settime(ships [3,0]) to: (clocktime + loading time);

timevalue (name)

provides the value of the time associated with the specified entity, *name. timevalue* is a function procedure.

scan (setname) for: (member) with: (leastvalue)

This is a special procedure for carrying out *phase-A* of the program. *setname* is a set containing entities to which

time values have been assigned. The procedure searches this set and selects the entity with the least time-value. *member* is an integer which at the end of the procedure is set equal to position 0 of the selected entity; in other words *member* is used in subsequent procedures as an implicit name for the selected entity. The parameter *leastvalue* is assigned the time-value of the selected entity. Thus, a typical series of statements would be as follows:

```
scan(timeset) for: (member) with: (leastvalue);
delete (member) from: (timeset);
clocktime:=leastvalue;
```

These statements select the entity with least time-value, delete that entity from the timeset, and advance simulation time to the time associated with the entity selected. If several entities in the set have the same *time-value = leastvalue,* then the first of these to appear in the set will be chosen.

DISTRIBUTION SAMPLING

random (number)

takes the next value from a stream of uniformly distributed random integers in the range 0 to 99. *number* specifies which of ten streams is to be used. A call with *number* taking the value zero reads a random number from the input medium specified by a global variable, *inputr.* Otherwise, *number* must be an integer in the range 1 to 10. In the latter case, a pseudo-random number will be generated by a multiplicative congruential method using one of the values in the integer array *zzxzz* [1:10] as a stream variable.

Note: For further information on the techniques of generating pseudo-random numbers, reference may be made to the following papers:

E.S. Page *"The generation of pseudo-random numbers."*
Computing Laboratory,
University of Newcastle Upon Tyne.

M.D. Maclaren *"Uniform Random Number Generators."*
and A.C.M. Journal Vol. 12 Number 1.
G. Marsaglia

Most textbooks on simulation also include one or more chapters on the generation of pseudo-random numbers.

distri (name, number)

reads in the co-ordinates of a cumulative probability distribution in the order: value, percentage probability, from the input channel specified by a global variable, *inputd.* There must be exactly 10 pairs of values for the percentage probabilities.

name must be an integer array with dimensions [0:20]. *number* must be in the range 0 to 10, and must also precede the distribution on the data tape. This number is used to check that the right data has been read for the required distribution. It also serves as a stream number when the distribution is sampled.

The first value of the percentage probability must be 0 and the last 100.

sample (distribution name)

takes a value sampled randomly from the specified distribution. This procedure calls the *random* procedure, using as stream number the value of *number* used as parameter to the *distri* procedure that created *distribution name.*

HISTOGRAMS

histogram (name, lowerbound, width)

sets up a histogram with the specified name. The histogram will have eleven zones; one below the value given by *lowerbound,* nine within consecutive *widths* above the lowerbound, and one above the highest zone width which will be *lowerbound* +9* *width. name* must be an integer array with dimensions [0:16].

addto (name, value)

adds one value to the named histogram. *name* is the histogram name. *value* is used to select the zone to be augmented.

writedown (name, title)

prints out the named histogram in tabular form preceded by *title.*

The parameter *title* must be a string, enclosed in string quotes. The mean, variance and standard deviation of the values in the histogram are also printed.

ADDITIONAL PROCEDURES IN ALSIMONA

The following procedures have been added by ICL to the set of procedures specified by the originator of SIMON, P.R. Hills.

primas

prints the contents of the master lists. This will consist only of numerical values, whose significance will not be apparent unless the user is familiar with the method of listing employed in the SIMON procedures. The *primas* procedure may, however, be of value in testing or debugging programs.

priset (setname)

prints the entries in the master lists which correspond to the set name, and the members of the specified set. *setname* must be an **integer** variable that has been defined as a list by using the SIMON procedure *set*.

prient (name)

prints the entries in the master lists corresponding to the named entity.

Examples:

```
prient (barber a [0]);
prient (ships [3,0]);
prient (balances [-3,0]);
prient (headof (timeset));
```

empty (setname)

empties a given set. *setname* must be an **integer** variable that has been defined as a list by using the SIMON procedure *set*.

belong (name, setname, marker)

sets *marker* equal to 1 if the named entity is a member of the specified set, otherwise sets it to -1. *marker* must be declared as an **integer** variable in the block in which it is used.

Example:

```
belong (headof (empty ships)) in : (atsea) maybe : (true);
```

ghist (name, lowerbound, n width)

declares *name* as a generalised histogram with *n* zones. One zone is between $-\infty$ and *lowerbound*, *n*-2 are of width *width*, and the last is between *lowerbound* + *(n-2)* **width* and $+\infty$.

name must be declared as an **integer array** dimensioned at least as *name* [0 : *n* + 5].

gaddto (name, value)

adds one value to the specified generalised histogram *name* which has been declared by *ghist*. *value* is used to select the zone to be augmented.

gwritehist (name, title)

is exactly analogous to *writedown* but is used when a generalised histogram has been declared by *ghist*.

Example:

```
integer array table [0:25]; integer i;
ghist (table, 1, 20, 1);
for i = 1 step 1 until 20 do gaddto (table, i);
gwritehist (table, (' table '));
```

ADDITIONAL PROCEDURES IN ALSIMONB

belonging (name) in: (setname)

is a **Boolean** procedure returning the value **true** if the named entity is a member of the specified set, otherwise returning the value **false.**

Example:

if *belonging (headof (empty ships)) in : (port said)* **then goto** *undock;*

addmiddle (name) to : (setname) at position : (n)

adds the named entity to the specified set at position *n.* If there is no position *n* (because there are too few members of the set), an error message is printed and there is a jump to the label *stop.*

Example:

addmiddle (s s pig [0]*) to : (empty ships) at position:* (3);

in position (number) in : (setname)

is a function procedure yielding an implicit name. (see *headof*). For example:

addfirst (in position (3) of : (empty ships)) to : (in port);

adds the third member of *empty ships* to the head of *in port,* leaving *empty ships* unchanged.

Chapter 3 Data

INPUT

The first item of data provided for any program written in SIMON must be a number which specifies the amount of storage to be allocated for the listing of entities and sets.

The various listing procedures used in SIMON operate on the elements of two one-dimensional arrays called *master lists.* Values are entered in these arrays corresponding to entities, set names, and members of sets, and these entries are linked together by various cross-references.

These arrays are declared and used automatically by the SIMON program, and need not concern the programmer, except that he must specify the number of elements in each of these arrays. This number may be determined from the formula:

$$sss = 2e + s + m$$

where *sss* is the value required; *e* is the number of entities defined in the program; *s* is the number of sets; and *m* is the maximum number of set members which may exist at any time.

If a particular entity may belong to several sets at the same time, it must be counted once for each of the sets, and once as an entity. For example, suppose a program defines 10 entities and 2 sets, and each of the entities may be added to both sets, then

$e = 10$

$s = 2$

$m = 20$

$sss = (2 \times 10) + 2 + 20 = 42$

If, on the other hand, each entity may only belong to one set at any time, then

$e = 10$

$s = 2$

$m = 10$

$sss = (2 \times 10) + 2 + 10 = 32$

The bulk of the data for simulation programs will usually consist of *probability distributions.* For each distribution, 21 numbers must be provided. The first number is an integer in the range 0 to 10 which identifies the distribution and also serves as a stream number when the distribution is sampled. The remaining 20 numbers are 10 pairs of values in the order: time, probability. The first value of probability must be 0 and the last 100, but neither time nor probability values need be evenly spaced. All values must be integers.

Example:

A typical set of data might be as follows:

50 (list size)

1 (identification/stream number)

Time	*Probability*
10	0
25	15
30	23
35	36

Time	*Probability*
36	48
37	67
38	78
40	95
45	98
60	100
2 (identification/stream number)	
180	0
195	13
	etc.

Distributions are read by means of the procedure *distri (name, n). name* is the name given to the distribution by the programmer; *n* is the identification number associated with the distribution. The number used in the call is compared with the number provided in the data, to check that the distribution read is the one intended by the programmer. For example, the above data might be read by means of the statements:

distri (short times, 1*);*

distri (long times, 2*);*

If these calls were made in the wrong order, however, or if the data was fed in the wrong order, an error message would be printed and the program would be halted.

OUTPUT

Output is handled by the normal Algol output procedures, except in the case of histograms which are output by the procedures *writedown (name, title)* and *gwritehist (name, title),* which tabulate the values in the histogram, and also calculate and print their mean, variance and standard deviation. *name* is the name of the histogram, and *title* is a string which is to be printed as a heading to the histogram. *title* must, of course, be enclosed in string quotes, and any spaces in the string must be represented by the appropriate symbol.

primas prints the master lists and can be used for monitoring, *prient* prints single entities and *priset* prints the contents of sets as integer values.

RANDOM NUMBERS

If random numbers are required in the program, the *random* procedure is used. This generates ten streams of numbers, and the parameter to the procedure is a number in the range 1 to 10 which selects the stream from which a number shall be taken.

The method used to generate these numbers in the 1900 version of SIMON is described in Chapter 5. If the standard *random* procedure is thought to be unsatisfactory for a particular application, the user may write his own procedure to generate random numbers.

Random numbers may also be prepared externally and read as data by the call *random* (0). Random numbers must be integers in the range 0 to 99.

Chapter 4 SIMON example

The following example shows the simulation of a barber's shop. The model is a simple one of a queue and two servers and is shown diagrammatically in Figure 1 on page 15. There are two barbers, *barber a* and *barber b* who each take some random time to cut a customer's hair; also, customers enter the shop at random intervals. *customer* is an entity representing all the customers in the queue at any one time, and the customer who is to arrive next. When a *customer* arrives he joins *queue*, if it contains less than four people; when a barber finishes a haircut, he joins *freebarbers*. *timeset* contains any customers about to arrive as well as any barbers who are currently engaged in cutting hair. The time at which events occur is recorded in *clocktime*.

It is assumed that if there are four or more customers waiting, the next *customer* refuses to join *queue*. When a *customer* either joins *queue* or refuses to join it, he is replaced by a further *customer*. The time when this next *customer* arrives is recorded and is subsequently compared with the time the next barber leaves *timeset* and joins *freebarbers*. The event with the earlier time is the next event when the program returns to *phase-A*, and is recorded in *leasttime*. *clocktime* is then advanced to *leasttime* in *phase-B*.

Input to the simulation is the distribution of arrival times for the customers, and the cutting times for the two barbers. The time for each barber to cut hair and the frequency of entry of customers is generated by randomly sampling three distributions referred to respectively in the program as *timebarb a*, *timebarb b*, and *timecust*. When used in conjunction with the procedure *distri*, they are followed by an integer which acts as a check that the correct distribution is being read in, and as a stream number for the random number generator used by the procedure *sample*.

The following data is used as input data for the distributions, consisting of respective times and probabilities.

Name	*timebarb a*		*timebarb b*		*timecust*	
Stream number	2		3		1	
Distributions	5	0	6	0	0	0
	8	11	6	11	0	11
	9	22	7	22	1	22
	9	33	9	33	1	33
	10	44	10	44	1	44
	10	55	10	55	2	55
	10	66	11	66	2	66
	10	77	12	77	3	77
	15	88	13	88	3	88
	18	100	20	100	4	100

When *timecust* is sampled, a random number is selected from Stream 1 and the appropriate time before the next customer enters the shop is found from the distribution; for example, if the random number were 76, the time value would be 2; if 77, the value would be 3. The random number selected, is always integer and in the range $0 \leqslant r < 100$, where r is the random number.

Output from the program takes two forms. A monitor print indicates the size of the queue, whenever either *freebarbers* or *queue* is empty or when a customer refuses to join the queue. The *clocktime* and size of queue at that time is printed. Whenever a customer refuses to join *queue*, the message CUSTOMER DOES NOT JOIN QUEUE is printed. A histogram, *queuesize*, is also printed at the end of the simulation run; this lists the frequency distribution of the size of the queue as shown in the monitor print. The mean, variance and standard deviation of the histogram are given.

The example program omits the listing of the SIMON procedures which precedes the actual simulation program, and starts with a call of the procedure *plantmaster*.

This is followed by an initialization phase which declares entities and sets, reads in distributions and set up the histogram *queuesize* which is later to be output.

The program then enters *phase-A*, *phase-B* and *phase-C*, as described on page 3. For clarity the beginning of each phase is labelled accordingly in the program. Whenever a return is made to *phase-A* during the simulation, a monitor print of the clocktime and the size of queue is made. The simulation is ended when the clocktime is equal to or greater than 34 units. The histogram is then printed. The sequence of events in the simulation is illustrated by the outline flowchart in Figure 2, on pages 16 and 17.

A list of the program and output, which should be self-explanatory to those familiar with Algol, is given beginning on page 18.

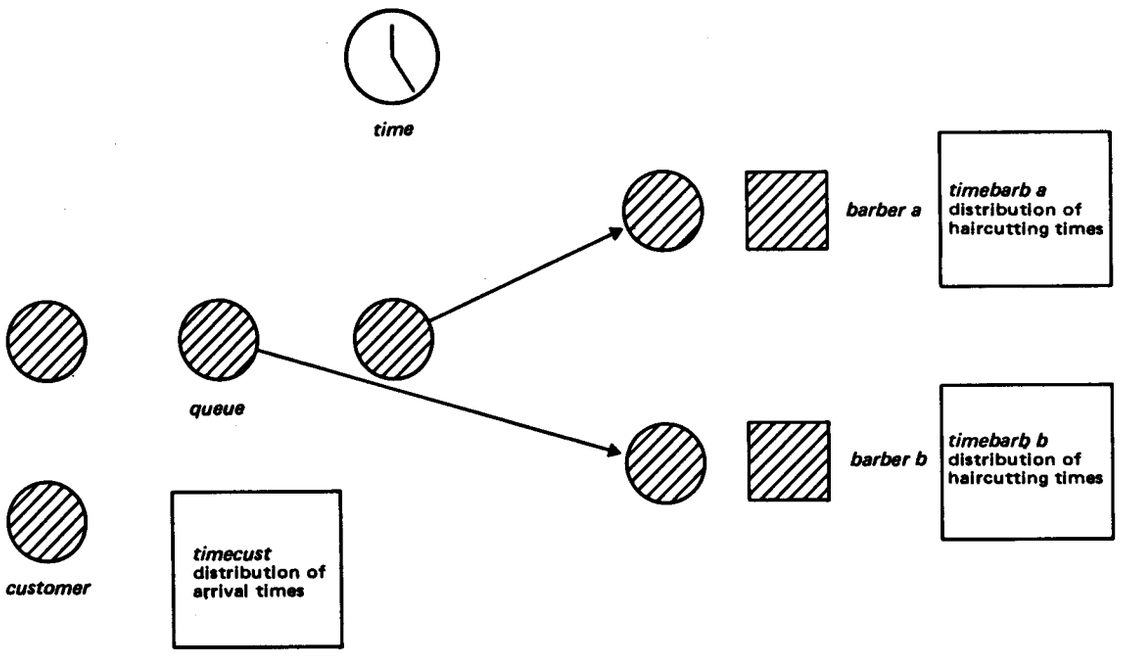


Figure 1 Simulation example

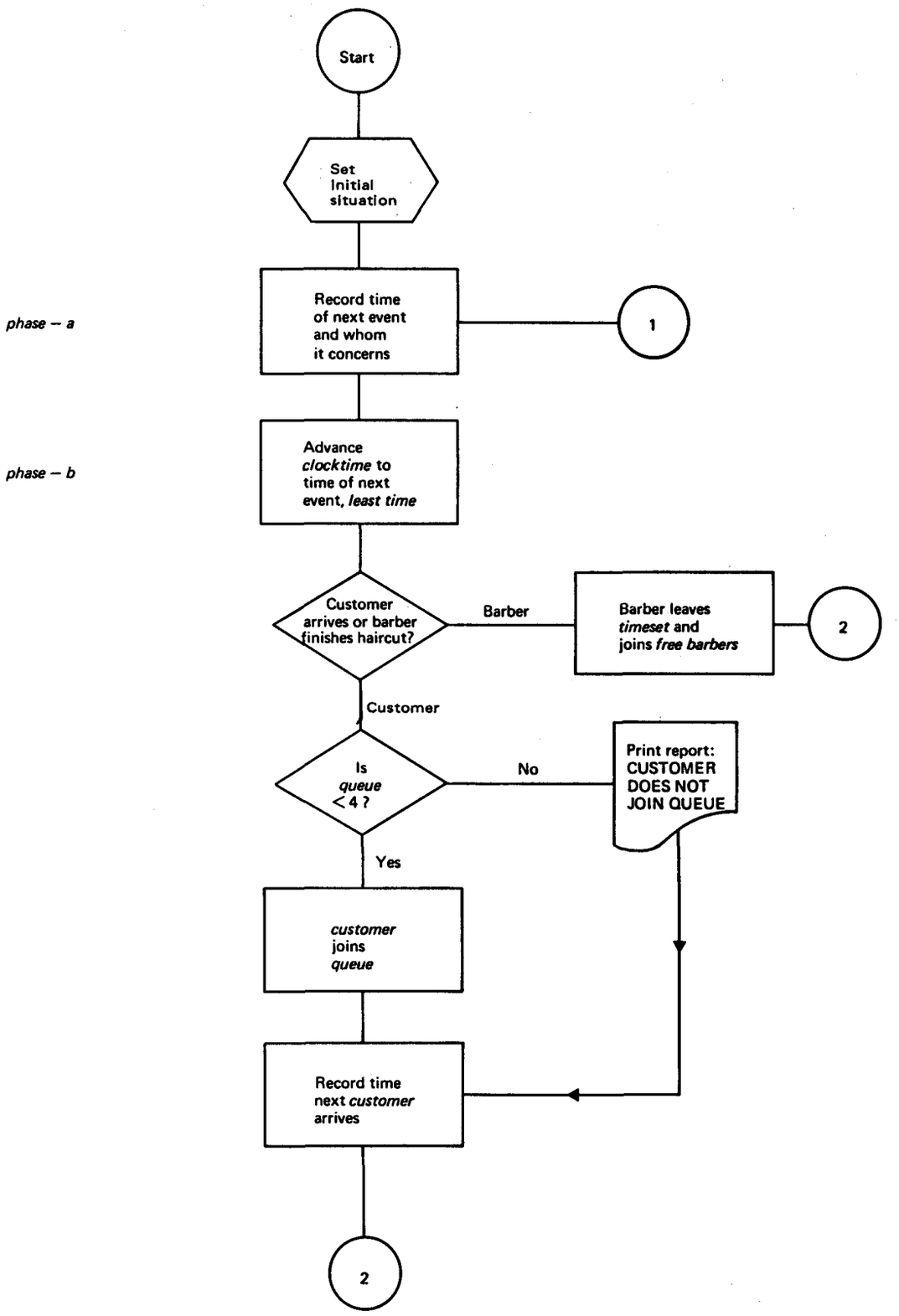


Figure 2 Outline flowchart

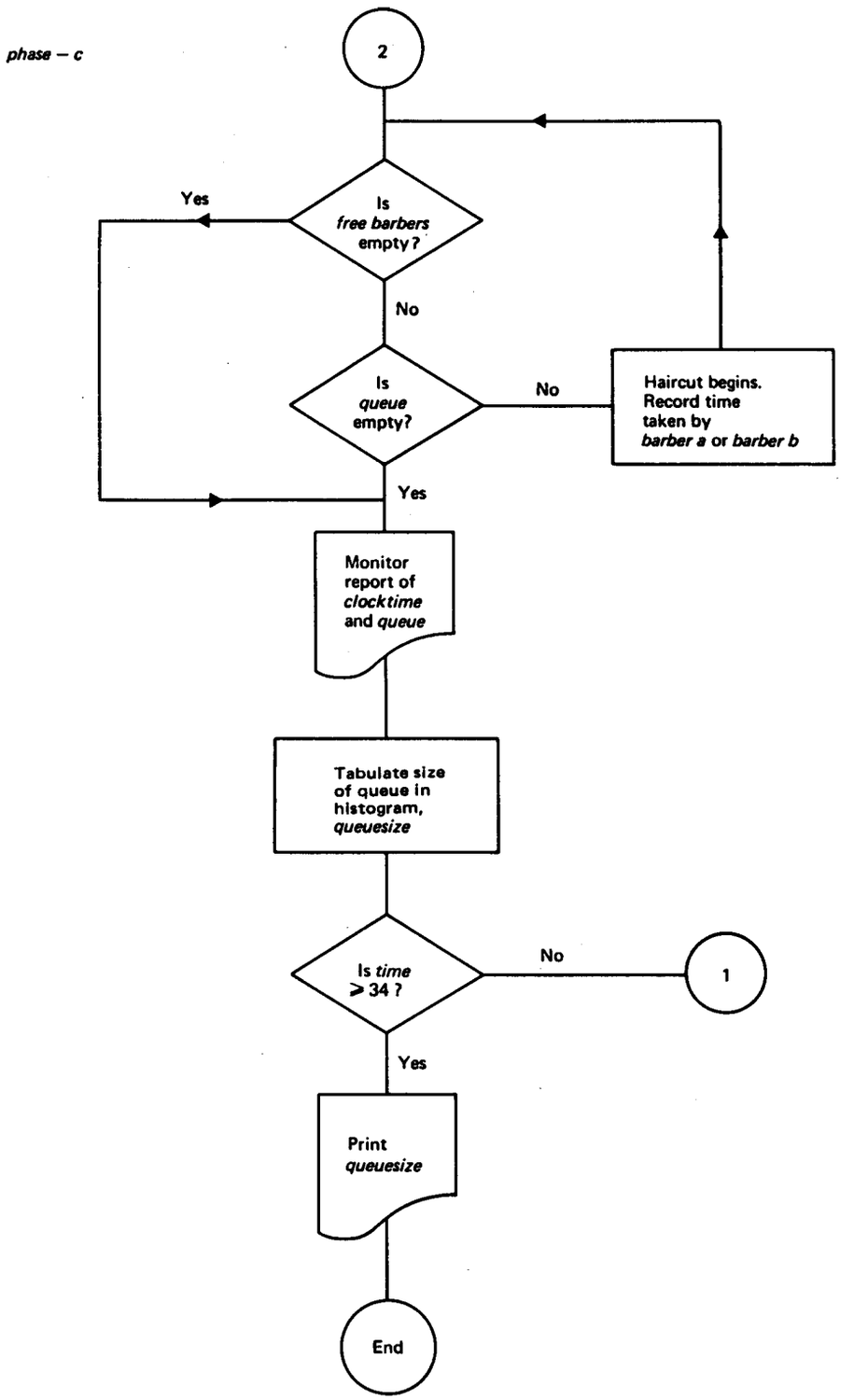


Figure 2 Outline flowchart continued

PROGRAM LISTING

The label *stop* and the extra **end** statements are for the SIMON procedures (not listed) which will precede the main simulation program.

```
comment start of user's program;
plantmaster;
for i: = 1 step 1 until 10 do zzxzz[i] : = i;
comment zzxzz is a global array for the 10 stream random number generator;
inputd: = 3;
comment 3 represents card reader input unit;
begin comment barbers shop with two assistants;
begin integer queue, freebarbers;
      integer timeset, leasttime, queue, clocktime, r, s, t;
      integer array customer, barber a, barber b [0 : 0];
                  timecust, timebarb a, timebarb b, queuesize [0 : 20];
comment print titles;
write text ("p'␣ ␣CLOCKTIME␣ ␣QUEUE');
comment define entities, sets and histogram, read in distribution;
entity (customer, 1); entity (barber a, 2); entity (barber b, 3);
set (freebarbers); set (queue); set (timeset);
distri (timecust, 1); distri (timebarb a, 2); distri (timebarb b, 3);
histogram (queuesize, 1, 1);
comment initiate simulation at time zero;
   begin
         addfirst (barber b [0], freebarbers);
         addfirst (barber a [0], freebarbers);
         settime (customer [0], sample (timecust));
         addfirst (customer [0], timeset);
         clocktime: = 0
   end;
comment enter phase a of simulation;
phase a : scan (timeset) for : (member) with : (leasttime);
comment enter phase b of simulation;
phase b : clocktime: = leasttime;
      if member = customer [0]
   then
   begin
     if sizeof (queue) < 4 then
     addlast (customer [0], queue) else
     begin writetext ('␣␣␣ ␣CUSTOMER␣DOES␣NOT␣JOIN␣ QUEUE');
        newline (2); end;
settime (customer [0], sample (timecust) + clocktime);
   goto phase c;
   end else
```

```
    begin
      addfirst (member, freebarbers);
      delete (member, timeset);
      goto phase c;
    end
comment enter phase c of simulation;
phase c : if size of (queue) > 0 ∧ size of (freebarbers) > 0
    then
    begin
      behead (queue);
      r: = headof (freebarbers);
      if r = barber a [0]
    then begin s: = sample (timebarb a);
                    settime (r, t + clocktime);
        end
    else begin t: = sample (timebarb b);
                    settime (r, s + clocktime);
        end;
    addfirst (headof (freebarbers), timeset);
    behead (freebarbers);
    goto phase c;
comment a rescan at same time to ensure that no more action can be carried out during phase c;
        end;
comment now phase c is finished the simulation will either terminate or return to phase a to advance the time to
          the point when the next event will take place;
comment print clocktime and size of queue;
print (clocktime, 8, 0); print (size of (queue), 8, 0);
newline (1);
addto (queuesize, size of (queue));
comment terminate if clocktime is greater than or equal to 34, else return to phase a;
if clocktime ≥ 34
then
    begin comment print the histogram and pause;
    writedown (queuesize, 'QUEUESIZE');
    pause (99);
        end;
goto phase a
end; end; end; stop: pause (99); newline (2); end
```

RESULTS

An output from one run of the program was as follows:

CLOCKTIME	QUEUE
0	0
1	1
4	2
5	1
6	0
7	1
7	2
9	3
10	4
11	3
12	2
12	3
15	4
CUSTOMER DOES NOT JOIN QUEUE	
19	4
19	3
19	2
20	3
23	4
CUSTOMER DOES NOT JOIN QUEUE	
23	4
CUSTOMER DOES NOT JOIN QUEUE	
23	4
CUSTOMER DOES NOT JOIN QUEUE	
24	4
CUSTOMER DOES NOT JOIN QUEUE	
26	4
29	3
29	4
31	3
33	4
CUSTOMER DOES NOT JOIN QUEUE	
34	4

QUEUESIZE	1	2	3	4	5	6	7	8	9	10	11
	2	3	4	7	11	0	0	0	0	0	0

MEAN	2.815
VARIANCE	1.632
STANDARD DEVIATION	1.278

Note that more than one event may occur at any one time, i.e. a customer entering the shop and a barber finishing cutting hair may coincide.

The histogram has the name QUEUESIZE; the first row of figures represents the frequency ranges of the queue size, the second the frequency. For example, if q is the queue size and f (q) the frequency of this queuesize appearing, the table is read as follows:

$0 \leq q < 1$	$f(q) = 2$
$1 \leq q < 2$	$f(q) = 3$
$2 \leq q < 3$	$f(q) = 4$
$3 \leq q < 4$	$f(q) = 7$
.	.
.	.
.	.

Of course, q is integer since fractional people cannot exist.

Underneath the histogram is printed its mean, variance, and standard deviation.

Chapter 5 Implementation on 1900 Series

SIMON PROGRAMS

The SIMON procedures are provided on paper-tape or cards. It is planned to add additional procedures from time to time, and a new SIMON program will be made available each time this is done. These packs or reels will be identified by the names ALSIMONA, ALSIMONB, ALSIMONC, and so on.

As each version is released a description of it will be issued, giving operating instructions and details of the additional facilities it provides.

LOCAL VARIABLES

The following variables are declared locally in the same block as the SIMON procedures:

nim
tailmost
mast 1
mast 2

They must not be used except by the SIMON procedures unless they are redefined in an inner block.

GLOBAL VARIABLES

Some variables are global; that is, they must be declared and assigned values in a block that encloses the block where the SIMON procedures are declared.

The names of these variables are as follows:

1 arrays: *zzxzz* [1:10]

zzxzz is the array from which the ten streams of random numbers are taken. It must be given initial values, which should preferably be large, say over 4,000,000.

2 integer variables: *inputr; inputd; sss;*

The variables *inputr* and *inputd* are used to define the channel numbers of the devices used to input data for the procedures *random (x)* (when $x = 0$) and *distri (a, n)*. *sss* is used to declare the size of the master lists, *mast 1* and *mast 2*.

GLOBAL LABEL

There is one label, *stop*, which is global; that is, it must appear in the block where the SIMON procedures are declared, or in a block enclosing that block. It must be provided by the user. *stop* precedes the error action to be taken after SIMON error states. The standard action recommended is

stop: primas; pause (99);

and in this case, as *primas* is one of the SIMON procedures, *stop* must be in the SIMON block.

ERROR STOPS

The 1900 SIMON program gives the following error messages:

MASTER LIST EXHAUSTED IN *procedure name*

The specified procedure cannot be carried out because there is no room for further entries in the master lists. This error is followed by a branch to the label *stop*.

SCAN EMPTY LIST

The *phase-A* scan procedure has been called when the set *timeset* is empty. This error is followed by a branch to the label *stop*.

MEMBER NOT PRESENT
The procedure *delete (name) from: (list)* has been called when the specified set does not contain the specified entity. The program will continue after this error.

HEADOF EMPTY LIST

TAILOF EMPTY LIST
headof or *tailof (list)* has been called when the specified list is empty. This error is followed by a branch to the label *stop.*

WRONG DATA NUMBER FOR DISTRIBUTION
The procedure *distri (name, n)* has been called, but the first number read in is not equal to *n.* This error is followed by a branch to the label *stop.*

RANDOM NUMBER GENERATOR
The *random* procedure generates numbers in the sequence given by the recurrence formula:

$$U_{i+1} = (3U_i) \text{ Mod } 2^{23} \qquad (i = 0, 1, 2 \ldots\ldots\ldots n)$$

The starting number U_0 is arbitrarily chosen and the result is truncated at both ends to provide a two-digit random number.

Chapter 6 Preparing a run using ALSIMONB

This chapter describes the application of the SIMON procedures provided by ALSIMONB, and lists the input for a simulation run.

The SIMON procedures are provided in Algol source language. A procedure *gener* for generating pseudo-random numbers is provided in semi-compiled form. This procedure is used, and declared as an **external integer** procedure in *random*.

The ALSIMONB pack of SIMON procedures as issued contains the SIMON procedures in alphabetical order preceded by:

```
begin
    integer nim, tailmost;
    integer array mast 1, mast 2 [0:sss-1];
```

SEQUENCE OF INPUT

The sequence of input in order to run a simulation program using ALSIMONB should be as follows:

Program description (including 'CONTINUE' to enable *gener* to be read).

```
begin integer array zzxzz [1:10];
    integer sss, inputr, inputd;
```

Statements to assign values to these variables, for example:

```
for I: = 1 step 5 until 50 do zzxzz[I] : = 8000000 - I;
sss: = 1000;
inputr: = 1;
inputd: = 3;
```

The ALSIMONB pack of SIMON procedures

User's own simulation program, which can start in the same block as the SIMON procedures.

```
stop: primas; pause (99);
end (to match begin at the start of the ALSIMONB pack;)

end of program;
```

The *gener* procedure is then read from tape or cards by means of a 'READ FROM' statement.

finish

Appendix 1 Summary of SIMON procedures

This appendix summarises the SIMON procedures described in Chapter 2. The format of each procedure is given, followed by a definition of the appropriate parameters. Explanatory remarks concerning the parameters are also included, but reference should be made to Chapter 2 for full details.

Note: throughout the appendix *list* always refers to a set name defined by *set*. Unless otherwise specified, *name* refers to an entity previously declared by *entity* or *groupentity*.

Procedure call	*Parameters*	*Remarks*	*Page*
addfirst (name, list)	**integer** *name, list;*	*name:* entity or set name	6
addlast (name, list)	**integer** *name, list;*	*name:* entity or set name	6
addmiddle (name, list, n)	**integer** *name, list, n;*	*name:* entity or set name *n:* position in specified set	10
addto (name, value)	**integer array** *name;* **integer** *value;*	*name:* histogram name defined by *histogram* *value:* value added to histogram	8
behead (list)	**integer** *list;*		6
belong (name, list, marker)	**integer** *name, list, marker;*	*marker:* takes value 1 or -1 provided by procedure	9
belonging (name, list)	**integer** *name, list;*	**Boolean procedure**	10
betail (list)	**integer** *list;*		6
delete (name, list)	**integer** *name, list;*	*name:* entity or set name	6
distri (name, n)	**integer array** *name;* **integer** *n;*	*name:* identification stream number *n:* in range 0 to 10	8
empty (list)	**integer** *list;*		9
entity (name, reference number)	**integer array** *name;* **integer** *reference number;*	*reference number:* chosen by programmer	5
gaddto (name, value)	**integer array** *name;* **integer** *value;*	*name:* histogram name defined by *ghist*	9
ghist (name, lowerbound, n, width)	**integer array** *name;* **integer** *lowerbound, n, width;*	*name:* histogram name *lowerbound:* lower limit of histogram zones *n:* number of zones *width:* zone width	9
groupentity (name, n, reference number)	**integer array** *name;* **integer** *n, reference number;*	*name:* name of groupentity *n:* number of members *reference number:* chosen by programmer	5
gwritehist (name, title)	**integer array** *name;* **string** *title;*	*name:* histogram name defined by *ghist* *title:* printed above histogram	9
headof (list)	**integer** *list;*	**integer procedure**	6

Procedure call	*Parameters*	*Remarks*	*Page*
histogram (name, lowerbound, width)	**integer array** *name;* **integer** *lowerbound, width;*	*name:* histogram name *lowerbound:* lower limit of histogram zones *width:* zone width	8
in position (n, list)	**integer** *n, list;*	**integer procedure** yielding an implicit name *n:* position of entity in set defined by *list*	10
memnum (name)	**integer** *name;*	**integer procedure** *name:* implicit name referring to member of a group entity	7
plantmaster		Sets up master lists	5
prient (name)	**integer** *name;*		9
primas		Prints contents of master lists	9
priset (list)	**integer** *list;*		9
random (n)	**integer** *n;*	*n:* stream number in range 0 to 10	8
refnum (name)	**integer** *name;*	**integer procedure** *name:* implicit name referring to an entity	7
rotate (list, n)	**integer** *list, n;*	*n:* number of rotations	6
sample (name)	**integer array** *name;*	**integer procedure** *name:* distribution name defined by *distri*	8
scan (list, member, leastvalue)	**integer** *list, member, leastvalue;*	*member:* entity in *list* with least timevalue assigned to *member* and its time value assigned to *leastvalue.*	7
set (list)	**integer** *list;*		5
settime (name, value)	**integer** *name, value;*	*value:* time value of specified entity	7
sizeof (list)	**integer** *list;*	**integer procedure**	7
tailof (list)	**integer** *list;*	**integer procedure**	6
timevalue (name)	**integer** *name;*	**integer procedure** *name:* entity assigned a time value by *settime*	7
writedown (name, title)	**integer array** *name;* **string** *title;*	*name:* histogram name defined by *histogram* *title:* printed above histogram	8

OXFORD UNIVERSITY COMPUTING LABORATORY
Copy 1.
COMPUTING SERVICE
4138

Amendment record

Should any additions or amendments to this Manual be necessary, they will be issued in the form of Amendment Lists.
When such changes have been incorporated, an appropriate entry should be made in the table below and the pink instruction sheet(s) should be filed behind this page.

List number	Item number
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to
	to