ELLIOTT 900

Volume 2: PROGRAMMING INFORMATION

Part 6: APPLIED PROGRAMMING

Section 3: ESP 900 (ALGOL SIMULATION PROCEDURES)

Contents

(Issue 2)

Printed at The Broadwater Press, Welwyn Garden City, Hertfordshire

Chapter 1: INTRODUCTION

The Elliott Simulator Package (ESP) is a set of ALGOL procedures designed to aid the writing of simulation programs.

This paper describes a version of ESP for use on the Elliott 900 data processing system.

The behaviour of complex systems, involving many interrelated processes, cannot usually be studied by normal mathematical techniques, but is most conveniently described by means of a model. The model once set up may be exposed to random demands and statistics collected to describe its response. This is the technique of simulation. The manipulation of the model usually involves only simple steps, but a great many steps are required if the results are to be statistically significant. Thus a numerical model, which can be manipulated by a computer, is usually devised.

One might question the need for special programming systems, such as ESP, for simulation studies, when powerful general-purpose systems such as ALGOL and FORTRAN are available. The answer is that the central difficulty of the problem is the control of the sequence in which the inter-dependant activities forming the model occur. If one attempts to write a simulation program using only a general-purpose language, one rapidly becomes enmeshed in the complexities of this sequencing control, which is not of great interest but nevertheless affords surprisingly fertile ground for minor errors. Moreover, mistakes here are liable to produce obscure effects, and are correspondingly difficult to eradicate.

The major task of a simulator programming system, then, is to deal with this sequencing problem. In addition, the user will need to program certain parts of his problem himself, and should be allowed to use some problem-orientated language. In the case of ESP this language is ALGOL. Finally, there are procedures which may be required by almost every user, such as random number routines, random sampling procedures and histogram procedures, and these too are built into the system.

Chapter 2: COMPUTER CONFIGURATION

ESP 900 is designed for an Elliott 900, data processing system with:

(a) At least 8, 192 word of Core store

(b) 1 Paper Tape Reader

(c) 1 Paper Tape Punch

An On-line teleprinter would be an advantage but not an absolute necessity.

A facility is included whereby the user may take advantage of an extra core store unit.

It is suggested that to obtain the maximum amount of space for the user's program it would be advisable to compile only those procedures which are required for the particular simulation. Further saving in store space may be achieved by the user performing a library scan (Volume 2.3.2 chapter 35).

N.B. SQRT is used by some of the ESP procedures.

Chapter 3: STRUCTURE OF THE SIMULATION MODEL

The system, of necessity, imposes certain constraints because of its form as a set of prefabricated units. Thus there is a preferred type of model, but the type is general enough to embrace almost all systems of interest and is self-explanatory enough to be easily written, explained and understood by others.

The system to be simulated may be regarded as a set of "actions" which manipulate a set of "objects". The objects might be ships waiting to be loaded by cranes, or the cargoes with which they depart. The actions in this case would be the arrival and departure of ships, and the beginnings and ends of loadings. Alternatively the objects may be cars being manufactured on a production line; and the actions the arrival and departure of the cars at certain points in the line and the starting and finishing of the jobs to be performed at those points. In the model, all objects are represented by numbers - or possibly by lists of numbers - and all actions are represented by units or blocks of program which manipulate the numbers.

There are two types of actions. There are those which are pre-ordained to occur at a certain time or after a set interval, which are known as "delayed actions", and there are actions which will occur as soon as some favourable combination of objects appears, and these are termed "conditional actions".

It is, of course, quite likely that hybrid actions could occur which depend on several conditions and also on time. Fortunately, hybrid actions can always be separated into a delayed action and a conditional action by making one of the conditions of the conditional action the fact that the delayed action has occurred and been noticed.

These concepts may be illustrated by an example. Let us consider empty delivery trucks arriving at a warehouse to be loaded. There are several loading bays, and a truck may be loaded at any bay. The objects of interest are free loading bays, waiting trucks and loaded trucks.

The actions are:-

(1) Whenever there is a free loading bay, and a waiting empty truck then the loading bay becomes busy and the truck disappears.

(2) After some time the loading bay becomes free and a loaded truck appears.

The first action is a conditional action which occurs whenever the stated objects are present. The second action is a delayed action which occurs at a pre-ordained time. This time was pre-set by action (1) when the loading commenced. The process by which a future occurrence of a delayed action is arranged will be referred to as "calling" that action.

It is the task of ESP to enter the various sections of the program representing actions in the correct sequence, and at the beginning of each adjust the value of a variable representing time. Any section entered may call any other section or itself. Provision is made for this and the subsequent behaviour of the sequencing routine will be modified in response to such calls.

Chapter 4: STRUCTURE OF THE SIMULATION PROGRAM

The program written by the user to perform his simulation will have a structure closely related to that of the model described in the previous section. The program will consist of several sections or blocks, each devoted to one action. It will be preceded by declarations of the identifiers and arrays needed to represent the objects to be manipulated, and to accumulate and record the results of the simulation. It must also have a block which sets up the initial state of the model. This block will normally be used only once, at the beginning of the simulation. The whole program is then embedded in an outer block containing the standard procedures of ESP so that these may be available anywhere within the program.

The various action blocks must be obeyed in proper sequence. Hence the routines of ESP first examine a list of actions which have been called but not yet employed, select the earliest of these and enter it. When this action has been completed, the remaining called actions are examined, and any which are called for at the same time as the one just performed are also entered. After this the conditional actions are examined, and any possible ones performed. The cycle is then repeated.

Since only the writer of the simulation can decide what tests are needed in the conditional actions, these tests are his responsibility. The whole set of conditional actions is grouped into one block. Within this block each action is represented by a conditional statement, e.g. in our example of the loading bays and trucks, we may have a conditional statement meaning : if a loading bay is free and there is an empty truck waiting, then the loading bay becomes busy and the number of empty trucks waiting is reduced by one. It is possible to group all the conditional actions together, because they must all be examined at the same time. There is thus no need to break into the middle of the group.

To enable ESP to enter the sections in the appropriate sequence, the starting point of each section is given a label. All these labels must be declared in a switch list, with the label corresponding to the conditional actions section first, followed by the labels corresponding to the delayed activities. The switch itself must have a name and for this description we shall use the name ACT. Thus if there are three delayed actions labelled, FIRST, SECOND and THIRD and the conditional happenings are labelled MAYBE, the switch declaration will be

"SWITCH" ACT:= MAYBE, FIRST, SECOND, THIRD;

Calls for actions are made through a procedure named CALL in ESP. This procedure has two parameters HAP and T, and is associated with the switch ACT as follows. The parameter HAP specifies the delayed action which is to be called and this is the (HAP)th delayed action in the

switch list, i. e. the (HAP + 1)th label. The conditional actions are never called but are entered automatically when needed. Thus in the list above CALL (2, T) is a call for the delayed action labelled SECOND. The parameter T specifies the time which is to elapse before the action actually occurs. Thus, if we want action FIRST to occur 1 time unit after the start of the simulation, and action THIRD to occur 1000 time units after the start, we write at the beginning of the program:

```
CALL (1, 1);
CALL (3, 1000);
```

Then if action SECOND is to occur 3 time units after action FIRST and action FIRST is to occur again every 6 time units, we write in the block which does everything concerned with action FIRST:

```
CALL (2, 3);
CALL (1, 6);
```

When a section of program has been performed, it is necessary to progress to the section representing the next action to be performed i. e. to jump to one of the labels in the switch ACT. ESP must decide which action is now required, and a procedure NEXT is provided for this purpose. This is an integer procedure and is used to select one of the labels from the switch ACT. Each section of the simulation program must end with the statement:

```
"GOTO"  ACT[NEXT];
```

This activates the procedure NEXT, which scans a list maintained inside ESP and selects the next action required. In this list, the actions are known by the parameter HAP given to the procedure CALL, and NEXT takes the appropriate value to cause entry to the section of program concerned. The procedure NEXT also causes entry to the group of tests for conditional actions. This occurs whenever a delayed action has just been completed, and the next delayed action is due to occur at a later time.

Thus the statement "GOTO" ACT [NEXT]; informs the simulation program that the current action is complete and another section is to be entered, whilst the statement CALL (HAP, T); is the means of pre-arranging future actions.

Chapter 5: PARAMETERS

It is quite likely that when an activity is called, certain data is available which will be useful when the action is performed. For example, suppose that the loading bays in our example are to be distinguishable, i.e. when a loading bay becomes busy we know which one it is, and similarly when it becomes free later we wish to recognise it as the same bay.

We might do this by having a large number of actions like "loading bay 1 becomes free", "loading bay 2 becomes free" and so on, but this is rather clumsy. Instead we prefer a single action "loading bay i becomes free" with a parameter i.

For the manipulation of parameters, two integer arrays are provided, declared within ESP. These are SEND [0:10] and GET [0:10]. Parameters to be transmitted are placed in SEND before calling for a delayed action. When the action occurs, it will find these same parameters in GET. The element SEND [0] is used for a special purpose - it states how many parameters are to be transmitted. If we wish to save certain information for use in a later block, for example a loading bay's number i, or some information about a truck, such as type of truck, goods to be loaded, driver to be allocated etc., we set this information in the array SEND before we set in motion the procedure CALL. We first set the number of parameters n, $1 \leqslant n \leqslant 10$, in SEND [0], and the parameters themselves in SEND [1] to SEND [n]. Procedure CALL will then store these parameters within its list of actions to be done, and reset SEND [0] to zero. If there are no parameters to be transmitted, it is unnecessary to clear SEND [0] before CALL. When an action having associated parameters is about to be entered through "GOTO" ACT [NEXT]; the parameters are automatically placed in the array GET and are available for use during the action. If the number of parameters, n, is less than 10, then GET [n + 1] to GET [10] are undefined. n is held in GET [0].

The ESP system makes it possible for several calls for future actions to exist concurrently. Thus several loadings may be in progress at any time with an associated call for "loading bay becomes free". The various times and parameters i are manipulated by ESP in such a way that entry occurs to the action at the appropriate times and at each entry the correct associated parameter i will be present. Thus the calls do not interfere with one another in any way.

Chapter 6: EXAMPLE 1

We can now consider an example program which uses the basic ideas of ESP introduced so far. Only the bare essentials of the simulation are considered and trivial statistics will be collected.

We will simulate a warehouse with four loading bays. Trucks are loaded for a period of 60 hours per week. They arrive at a rate of one every twenty minutes. Any truck may be loaded at any loading bay. It takes one hour to load a truck. We wish to find the number of trucks loaded at each bay during a 60 hour working week, assuming that the bays are numbered from one to four, and that if more than one bay is free then the lowest numbered bay accepts the next empty truck for loading.

The program is:

```
"BEGIN" "BOOLEAN" open;
"COMMENT" open is true for 3600 minutes, then false;
"INTEGER" truckswaiting;
"COMMENT" this is a count of the empty trucks waiting;
"BOOLEAN" "ARRAY" bayfree [1 : 4];
"COMMENT" bayfree [i] is true if bay i is free;
"INTEGER" "ARRAY" loadedtrucks [1 : 4];
"COMMENT" loadedtrucks [i] is a count of the trucks loaded
          by bay i;
"SWITCH" ACT:= beginload, truckarrive, finishload, shutwarehouse,
          results;
"BEGIN" "INTEGER" i;
"COMMENT" this is the program entry point and this section
          sets up the initial conditions of the system.
          PREPARE is an ESP procedure which initialises
          ESP;

PREPARE;

open:= "TRUE"; truckswaiting:=0;
"FOR" i:=1 "STEP" 1 "UNTIL" 4 "DO"
      "BEGIN" bayfree[i]:= "TRUE";
              loaded trucks[i]:=0
"END";

CALL (1, 10);
"COMMENT" the first empty truck arrives in 10 minutes;
CALL (3, 3600);
"COMMENT" the warehouse closes in 3600 minutes;
CALL (4, 4000);
"COMMENT" output results after 4000 minutes, i.e. when all
          other activity has ceased;
```

```
"GOTO" ACT [NEXT]
"END"     This completes the setting up of the initial state. The
          procedure NEXT will cause entry to the appropriate
          section of program. On this occasion, this will be the
          section labelled 'truckarrive', in response to CALL
          (1, 10);

truckarrive:  "BEGIN" "COMMENT" this section simulates
              the arrival of an empty truck. It also calls itself
              so that a further truck will arive in due course.
              The process stops when 'open' becomes false;
              "IF" open "THEN" "BEGIN"
              truckswaiting:=truckswaiting +1;
              CALL (1, 20)
              "END";
"GOTO" ACT [NEXT]
"END" on this first occasion, NEXT will select the conditional action
'beginload';

beginload:    "BEGIN" "COMMENT" this is the set of conditional
              actions which starts loading a truck whenever a
              loading bay and an empty truck are both available.

"INTEGER" i;
"FOR" i:=1 "STEP" 1 "UNTIL" 4 "DO"
"IF" bayfree[i]"AND" truckswaiting "NE" 0 "THEN"
"BEGIN" bayfree[i]:="FALSE";
          truckswaiting:= truckswaiting - 1;
          "COMMENT" the loading bay is now busy and we must
                    make it become free again in 1 hour;
          SEND [0]:=1;SEND [1]:=i;
          CALL (2, 60);
          "COMMENT" we need to transmit the loading bay's
                    number i, to enable us to make the
                    correct loading bay become free;
          loaded trucks [i]:= loaded trucks [i] + 1
"END";
"GOTO" ACT [NEXT]
"END"     the loading thus started will be terminated by the action
          labelled 'finishload';
finishload: bayfree [GET [1]]:= "TRUE";
          "COMMENT"  this action frees the bay after the loading
                     started in 'beginload' has finished. The
                     bay's number i is in GET [1], corresponding
                     to the SEND [1] of the action 'beginload';
          "GOTO" ACT [NEXT];
```

```
shutwarehouse: open:= "FALSE";
          "COMMENT" this action prevents the arrival of further
                    empty trucks since 'open' is tested by the
                    action 'truckarrive';
          "GOTO" ACT [NEXT];
results:  "BEGIN" "INTEGER" i;
          "FOR" i:=1 "STEP" 1 "UNTIL 4 "DO"
          "PRINT" loadedtrucks [i];
          STOP;
          "END" there will not be any more called events at
                this stage, ESP will output a message to this
                effect and stop.
          "END" of example program which must be followed by
          "END"
          "END" to close the surrounding blocks containing ESP
                and its procedures;
```

As one might expect the results were:

```
Trucks loaded at bay 1  =  60
Trucks loaded at bay 2  =  60
Trucks loaded at bay 3  =  60
Trucks loaded at bay 4  =   0
```

Chapter 7: THE RANDOM NUMBER PROCEDURES

Example 1 is very simple and in many ways unrealistic. We said that empty trucks arrived once every twenty minutes, while in real life it is more likely that they arrived at varying intervals, the average length of which was twenty minutes. Equally, the loading time is more likely to average one hour. Thus we would prefer to have trucks arriving at random intervals with a mean time of twenty minutes. We would also like to have random numbers available for other purposes and to form separate streams of random numbers so that different parts of the program may be independent.

Several random number procedures are provided in ESP.

The process to generate a stream of numbers which may be regarded as random is described by A. R. Edmonds in the Computer Journal, Vol. 2, No. 4, Page 181 onwards. The random numbers u_1, u_2 ..... are generated by a series defined by

$$u_n + 1 = ku_n. \quad \text{modulo } m$$

The formula that ESP uses is

$$u_n + 1 = 13^{13}.\ u_n \quad \text{modulo } (2^{31}-1) \qquad -(1)$$

i. e. $m = 2^{31} - 1 = 2147483647$

and $k = 13^{13}$ modulo m = 455 470 314

In (1) the u_n form a periodic sequence of numbers with period $2^{31} - 2$. Any positive integer "a" may be used as the starting point of the sequence provided a<m. This is Lehmer's method.

We may conveniently generate several random number streams using (1) provided that the various starting values "a" are taken from widely separated points of the complete periodic sequence.

Now if $u_1 = a$

then $u_n = k^n a.$ modulo m

i. e. $u_n = ca.$ modulo m

where $c = k^n.$ modulo m.

Thus, for large n, the series a_1, a_2 ..... defined below will be a good set of starting values for the various random number streams.

$$a_1 = 28\ 747\ 135$$

$$a_i + 1 = ca_i. \quad \text{modulo } m \qquad -(2)$$

If $n = 2^{25}$ then $c = 1\ 071\ 581\ 425$. In this case 64 values of "a" from widely separated parts of the cycle will be given and a random number stream starting from any one of these will give 2^{25} values of u_n before overlapping the beginning of the next stream.

7.1 Double length Random Integers

In 900 Elliott Algol a variable declared to be of type real is allocated two locations. In this version of ESP the random number routines use these two locations to hold double length integers. This is because the random number generator (1) is ideally suited to a 31 bit word whereas the 900 data processing systems have 18 bit words.

Note, then, that any variables declared by the user, which are used as double length random integers, must be declared to be of type real.

7.2 "INTEGER" "PROCEDURE" BASERANDOM (U);

This procedure produces a series of numbers on successive entries such that all numbers between 1 and $2^{31} - 2$ are produced only once in the cycle and are equally likely to occur.

U, a real variable used to hold 31 bits random integers, is replaced by the next term in the sequence u_n. BASERANDOM itself takes a positive integer value corresponding to the top 17 bits of the new random integer.

Note: A real random number R in the range 0 to 1.0 may be obtained by the statement:

R: = BASERANDOM (U)/131071.0;

7.3 "REAL" "PROCEDURE" SETRANDOM;

This procedure is used to initialise any number of random number streams for use as actual parameters in BASERANDOM or the Random Sampling Procedures (ref. Chapter 8). These parameters must be declared to be of type real at the beginning of the user's program. The starting value of SETRANDOM is set by the procedure PREPARE, and subsequent calls will produce in succession the numbers a_1, a_2 ..... defined by (2) and these can then be assigned to the actual parameters.

Chapter 8: THE RANDOM SAMPLING PROCEDURES

All the random sampling procedures use one or more terms from the sequence u_n (ref. Chapter 7). They all require a parameter, U, the name of a real variable (i.e. double length integer, ref. Chapter 7.1) which is used to contain successive terms of the sequence u_n, and the value of which is changed each time the procedure is obeyed.

In general, random samples from continuous distributions are of type real while those from discrete distributions are of type integer.

8.1 "INTEGER" "PROCEDURE" RANDOM (A, B, U);

The result of this procedure is a random integer X such that $A \leq X \leq B$, i.e. a Rectangular distribution. The procedure obtains a random number R in the range (0, 1) via BASERANDOM and evaluates X by

X: = ENTIER (R* (B - A+1)) + A

A and B are of type integer.

8.2 "REAL" "PROCEDURE" NEGEXP (MEAN, U);

The resultant random number X is selected from the Negative Exponential distribution having mean MEAN (type real). The procedure obtains a random number R in the range (0, 1) via BASERANDOM and evaluates X by

X: = - MEAN * LN(R)

8.3 "REAL" "PROCEDURE" NORMAL
(MEAN, SIGMA, U);

The result of this procedure is a Normal random variable distributed with mean MEAN and standard deviation SIGMA. The procedure sums 12 random numbers in the range (0, 1) generated via BASERANDOM, and obtains X from

$$X: = MEAN + SIGMA*[(\sum_{1}^{12} R) - 6]$$

MEAN and SIGMA are of type real.

The procedure may be adapted to use less than 12 random numbers, but although such a version would be faster the result is liable to be biased. However, a procedure FASTNORMAL is provided.

8.4 "REAL" "PROCEDURE" FASTNORMAL
(MEAN, SIGMA, U);

This is similar to 8.3 but only 5 numbers are summed.

$$X: = MEAN + \sqrt{3/5} * SIGMA * [2*(\sum_{1}^{5} R) - 5]$$

8.5 "REAL" "PROCEDURE" LOGNORMAL
(MEAN, SIGMA, U);

The result X of this procedure is a random variable selected from a Lognormal distribution with mean MEAN and standard deviation SIGMA. MEAN and SIGMA are of type real.

The procedure sums 12 random numbers in the range (0, 1) generated via BASERANDOM, and evaluates X from

$$X := \text{EXP}\left(\text{MEAN} + \text{SIGMA} * \left[\left(\sum_{1}^{12} R\right) - 6\right]\right)$$

8.6 "INTEGER" "PROCEDURE" POISSON
(MEAN, U);

The resultant random integer X is selected from a Poisson distribution having mean and variance MEAN (type real). The procedure obtains a random number R in the range (0, 1) via BASERANDOM and evaluates X as follows:

Let m = MEAN

If $\exp(-m) \geq R$, then X = 0

If $\exp(-m)(1+m) \geq R > \exp(-m)$, then X = 1

If $\exp(-m)\left(1+m+\frac{m^2}{2!}\right) \geq R > \exp(-m)(1+m)$, then X = 2

etc.,

Note: $\exp(-m)\left(1+m+\frac{m^2}{2!} + \frac{m^3}{3!} + \ldots\ldots\right)$

$= \exp(-m) \cdot \exp(m) = 1$

8.7 "INTEGER" "PROCEDURE" BINOMIAL (N, P, U);

The resultant random integer X is selected from a Binomial distribution. P is the probability of a "success" and N is the number of "attempts". The proceudre obtains a random number R in the range (0, 1) via BASERANDOM and evaluates X as follows:

Let P = 1 - Q

If $Q^N \geq R$, then X = 0

If $Q^N < R \leq Q^N + NPQ^{N-1}$, then X = 1

If $Q^N + NPQ^{N-1} < R \leq Q^N + NPQ^{N-1} + N\frac{(N-1)}{2!} P^2 Q^{N-2}$

etc., then X = 2

Note: $Q^N + NPQ^{N-1} + N\frac{(N-1)}{2!} P^2 Q^{N-2} + \ldots\ldots + P^N = (Q+P)^N = 1.$

N is integer, P is real.

8.8 "INTEGER" "PROCEDURE" PASCAL (M, P, U);

If P is the probability of a "success" then the resultant random integer X is selected from a pascal distribution and represents the number of "attempts" necessary to obtain M "successes". The procedure obtains a random number R in the range (0, 1) via BASERANDOM and evaluates X as follows:

Let P = 1 - Q

If $P^M \geqslant R$, then X = M

If $P^M < R \leqslant P^M (1+MQ)$, then X = M+1

If $P^M (1+MQ) < R \leqslant P^M [1+MQ+\frac{M(M+1)}{2!} Q^2]$, then X = M+2

etc.,

Note 1: $P^M [1+MQ+M\frac{(M+1)}{2!} Q^2 + \ldots\ldots\ldots] P^M (1-Q)^{-M} = 1.$

Note 2: A random sample from a Geometric distribution may be obtained by calling PASCAL with M=1. However, a more efficient procedure GEOMETRIC is provided.

8.9 "INTEGER" "PROCEDURE" GEOMETRIC (P, U);

If P is the probability of a "success" then the resultant random integer X is selected from a Geometric distribution and represents the number of "attempts" that occur before a "failure" occurs.

The procedure obtains a random number of R in the range

(0, 1) via BASERANDOM and evaluates X as follows:

Let P = 1 - Q

If $P \geqslant R$, then X = 1

If $P < R \leqslant P (1+Q)$, then X = 2

If $P (1+Q) < R \leqslant P (1+Q+Q^2)$, then X = 3

etc.,

Note: $P (1+Q+Q^2+Q^3 \ldots\ldots) = P (1 - Q)^{-1} = 1.$

8.10 "REAL" "PROCEDURE" ERLANG (A, K, U);

The resultant random variable X is selected from an Erlang distribution. A and K are two parameters defined by:

$$A = \frac{\text{mean}}{\text{variance}} \qquad \text{Type } \underline{\text{real}}$$

$$K = \frac{(\text{mean})^2}{\text{variance}} \qquad \text{Type } \underline{\text{integer}}$$

The procedure obtains a series of random numbers R_i in the range (0, 1) via BASERANDOM and evaluates X from

$$X := -\frac{1}{A} * LN \left(\prod_{1}^{K} R_i\right)$$

8.11 "INTEGER" "PROCEDURE" HYPERGEOM (TN, NS, P, U);

The resultant random integer X is selected from a Hypergeometric distribution. TN is the size of a population consisting of class I and class II elements. P is the probability of any element in the population being class I. X will be the number of class I elements in a sample of NS elements selected randomly from the population without replacement.

The method is as follows:

(a) X is initially set to zero.

(b) A random number R in the range (0, 1) is generated via BASERANDOM.

(c) If $R \leqslant P$ it is taken to represent a class I element and X is increased by 1.

(d) The population size TN is decreased by 1 and the new value of P calculated.

(e) Steps (b), (c) and (d) are repeated NS times in all.

TN and NS are integers, P is real.

Chapter 9: EXAMPLE 2

To illustrate the use of the random number procedures we shall extend example 1. We now specify that the arrival times of empty trucks have a negative exponential distribution with a mean of twenty minutes and the loading time of the trucks is a random value uniformly distributed between 40 and 80 minutes. The general structure will remain the same, and the only blocks in our previous example which we must change are those labelled "truckarrive" and "beginload" and the preparatory block. We also need two more real variables, u1 and u2, to be source parameters for the random number routines.

Our Program now becomes, missing out some of the comments which apply as before.

```
"BEGIN" "BOOLEAN" open; "INTEGER" truckswaiting;
        "REAL" u1, u2;
        "BOOLEAN" "ARRAY" bayfree [1 : 4];
        "INTEGER" "ARRAY" loadedtrucks [1 : 4];
        "SWITCH" ACT:= beginload, truckarrive, finishload,
                       shutwarehouse, results;
        "BEGIN" "INTEGER" i;
        "COMMENT" new initialising section;
                  PREPARE;
                  open:= "TRUE"; truckswaiting:=0;
                  "FOR" i:=1 "STEP" 1 "UNTIL" 4 "DO"
                  "BEGIN" bayfree [i] := "TRUE";
                          loadedtrucks [i] :=0
                  "END";
                  u1:= SETRANDOM; u2:= SETRANDOM;
        "COMMENT" initialises two random number streams;
                  CALL (1, 10); CALL (3, 3600);
                  CALL (4, 4000);
        "GOTO" ACT [NEXT]

        "END";

truckarrive:  "BEGIN" "IF" open "THEN"
              "BEGIN" truckswaiting := truckswaiting +1;
                      CALL (1, NEGEXP (20, u1);
              "COMMENT" we now simulate the random
                        arrival of an empty truck;
              "END";
              "GOTO" ACT [NEXT]
              "END";
```

```
beginload:      "BEGIN" "INTEGER" i;
                    "FOR" i:=1 "STEP" 1 "UNTIL" 4 "DO"
                    "IF" bayfree[i]"AND" truckswaiting
                                "NE" 0 "THEN"
                    "BEGIN" bayfree[i]:= "FALSE";
                            truckswaiting:=truckswaiting -1;
                            SEND [0]:=1; SEND [1] :=i;
                            CALL (2, RANDOM (40, 80, u2);
                    "COMMENT" we thus make the loading time
                                of random length;
                                loadedtrucks [i] := loadedtrucks [i] + 1
                    "END";
                    "GOTO" ACT [NEXT]
                "END";
finishload:     bayfree [GET [1]] := "TRUE";
                "GOTO" ACT [NEXT];
shutwarehouse:  open:= "FALSE";
                  "GOTO" ACT [NEXT];
results:    "BEGIN" "INTEGER" i;
                "FOR" i :=1 "STEP" 1 "UNTIL" 4 "DO"
                "PRINT" loadedtrucks [i];
                STOP
            "END"
            "END" user's program
            "END"
            "END" ESP;
```

The results are as follows:

```
Trucks loaded at bay 1 = 49
Trucks loaded at bay 2 = 49
Trucks loaded at bay 3 = 44
Trucks loaded at bay 4 = 36
```

Chapter 10: THE HISTOGRAM PROCEDURES

Facilities are provided for accumulating histograms, for performing various operations on them and for sampling from distributions of observed data.

A histogram consists of a set of numbers, each number being the frequency of occurrence of a value of a variate in some particular interval. Histograms used with ESP may comprise any number of cells; the cells being adjacent intervals of equal size. The interval size, or cell width, must be integer and the cell boundaries are regarded as lying between two successive integers. Only integer values are allowed so that there is no doubt as to the correct cell in which a score is to be made.

10.1 Histogram Structure

Each histogram occupies a one-dimensional integer array whose lower bound is always -7. Most of the array elements are used to store frequency counts for particular cells. Others, which are in the same position for every histogram, store the parameters of the histogram i.e. number of cells, cell width, etc. In addition, frequency counts are kept for values outside the range of the cells, one for those values over the upper limit and another for those below the lower limit. Running totals, $\sum x$ and $\sum x^2$, are kept so that the true mean and variance may be computed, and a count is kept of the total number of observations.

For a histogram of N cells, an array of N + 8 elements is required with lower subscript bound -7 and upper subscript bound N.

The N + 8 elements are used as follows:

Subscript	Use	
- 7	Number of cells, :N	Parameters
- 6	Lower bound of cell 1, :L	Parameters
- 5	Cell width, :W	Parameters
- 4	Number of observations, n	Running totals
- 3	Sum of values of observations, $\sum x$	Running totals
- 2	Sum of squares of values, $\sum x^2$	Running totals
- 1	Frequency count "UNDER"	
0	Frequency count cell 1	
1	Frequency count cell 2	
2	Frequency count cell 3	
.		
.		
.		
.		
N-1	Frequency count cell N	
N	Frequency count "OVER"	

10.2 "PROCEDURE" HSTSET (A, N, L, W);

The storage area for a histogram is reserved by declaring an integer array with the appropriate subscript bounds and then using the procedure

HSTSET (A, N, L, W);

where N, L, W are defined above, and A is the name of the histogram array.

For example, "INTEGER" "ARRAY" EXAMPLE [-7:4];
HSTSET (EXAMPLE, 4, , 1, 3);

will set up an array called EXAMPLE with cells:-

"UNDER"	0 and below
cell 1	1 to 3
cell 2	4 to 6
cell 3	7 to 9
cell 4	10 to 12
"OVER"	13 and above

The upper subscript bound may be greater than N; in which case the extra elements are not disturbed. The procedure then stores N, L and W in their appropriate positions, and clears the remainder of the histogram.

10.3 "PROCEDURE" HSTPUT (A, X);

To insert a score in the appropriate cell of a histogram the procedure

HSTPUT (A, X);

is called, where A is the name of the histogram array and X the value to be inserted. Thus the statement

HSTPUT (EXAMPLE, 7);

will cause the count in cell 3 of the histogram EXAMPLE to be increased by 1, and the number of observations, sum of values and sum of squares to be adjusted correspondingly.

10.4 "PROCEDURE" HSTPRINT (A, DEVOUT);

The contents of the cells of a histogram may be output at any stage by calling the procedure

HSTPRINT (A, DEVOUT);

where A is the name of the histogram.

DEVOUT is an integer variable which determines the output device as follows:-

DEVOUT=1	output is on the punch
DEVOUT=3	output is on the teleprinter

The contents of each cell are identified and printed on a new line followed by the proportion of values falling in each cell and the cumulative proportion. Thus a call of

HSTPRINT (EXAMPLE, 3);

might produce the following output on the on-line teleprinter.

UNDER	4	.11428571	.11428571
1 : 3	7	.20000000	.31428571
4 : 6	12	.34285714	.65714286
7 : 9	9	.25714286	.91428571
10 : 12	1	.05714286	.97142857
OVER	1	.02857143	1.0000000

10.5 "PROCEDURE" HSTTRUE (A, M, V);

The true or ungrouped mean and variance may be obtained by using the procedure

HSTTRUE (A, M, V);

where A is the histogram array, and M and V the resulting mean and variance. M and V are of type real.

10.6 "PROCEDURE" HSTGROUP (A, M, V);

The grouped mean and variance may be obtained by the procedure

HSTGROUP (A, M, V);

with A, M, V defined as in 10.5.

The formulae used are:-

$$M = \frac{1}{n} \sum_{i=0}^{N-1} (L + iW) \,.\, A[i] + \tfrac{1}{2}(W-1)$$

$$= L + \tfrac{1}{2}(W-1) + \frac{W}{n} \sum_{i=1}^{N-1} i.A[i]$$

$$V = \frac{1}{n-1} \left\{ \sum_{i=0}^{N-1} (L + iW)^2 \,.\, A[i] - \frac{1}{n} \left\{ \sum_{i=0}^{N-1} (L+iW) \,.\, A[i] \right\}^2 \right\}$$

$$= \frac{W^2}{n-1} \left\{ \sum_{i=1}^{N-1} i^2 \,.\, A[i] - \frac{1}{n} \left\{ \sum_{i=1}^{N-1} i \,.\, A[i] \right\}^2 \right\}$$

where n is the total number of observations.

The formula for the mean assumes that the values represented by a count in a cell, lie at the centre of that cell.

The formula for the variance gives an estimate of the variance of the population from which the histogram has been drawn, rather than the histogram itself.

If there are any counts in the "UNDER" or "OVER" cells then the mean and variance are undefined. The procedure then exits with M = largest positive real number and V = -1.

10.7 Arithmetic Operations

Simple arithmetic operations may be performed on histograms provided that if more than one histogram is involved, they must all have the same values of N, L and W.

There are three permitted operations viz addition, subtraction and multiplication by a constant.

The available procedures are:-

```
HSTADD (A, B, C);
HSTSUB (A, B, C);
HSTMULT (A, K, C);
```

Thus if a, b, c, represent cells of A, B, C respectively, then the results of the procedures are:

$c = a + b$	
$c = a - b$	where $a \geq b$
$c = Ka$	where K is a positive integer constant.

In the procedures B and C may both be equal to A or to each other. A copy of a histogram may be made by calling HSTMULT with K = 1.

All the procedures check that the histograms are compatible and that no negative counts are produced. In the event of either of these occurring ESP will output an error message and stop.

10.8 "PROCEDURE" HSTIN (A);

Histograms may be input ready for use by the procedure

```
HSTIN (A);
```

where A is an array of sufficient size and with lower subscript bound - 7. The input tape must contain N+8 numbers. The numbers representing the number of observations, sum of values and sum of squares may <u>all</u> be input as zero or else they must <u>all</u> be given values. In the former case the missing quantities will be constructed from the remaining data, though the actual sums of values and sums of squares will be replaced by grouped sums.

The number of observations, if given, is checked against the sum of the cell counts.

10.9 "PROCEDURE" HSTOUT (A);

A histogram may be output in a form suitable for re-input by HSTIN (A) by the procedure

```
HSTOUT (A);
```

The values are output in ascending order of subscript.

10.10 "INTEGER" "PROCEDURE" HSTSAMP (A, U);

A histogram may be used as a distribution and sampled by using the integer procedure

HSTSAMP (A, U);

where U is a real variable used to hold a series of random numbers (ref. 7.1).

The result of this procedure is an integer X such that the probability that X lies in any cell of the histogram is proportional to the count in that cell. If the cell width exceeds unity, then all integer values of X which would lie in the given cell are equally likely. If X lies in "UNDER" or "OVER" then X will be given the value of the cell boundary. Thus in the histogram EXAMPLE (ref. 10.2) the value of X obtained from "UNDER" would be 0 while 13 would be obtained from "OVER".

Chapter 11: SUMMARY OF PROCEDURES

The following is a table of available procedures. The user is recommended to omit those not required when running the simulation program.

Procedure	Type	Remarks	approx. store used
CALL (HAP, T); NEXT; PREPARE;	- "INTEGER" -	These are part of the Housekeeping Section and must always be included	750
BASERANDOM (U); SETRANDOM;	"INTEGER" "REAL"	The Random Number Procedures	20
RANDOM (A, B, U); NEGEXP (MEAN, U); NORMAL (MEAN, SIGMA, U); FASTNORMAL (MEAN, SIGMA, U); LOGNORMAL (MEAN, SIGMA, U); POISSON (MEAN, U); BINOMINAL (N, P, U); PASCAL (M, P, U); GEOMETRIC (P, U); ERLANG (A, K, U); HYPERGEOM (TN, NS, P, U);	"INTEGER" "REAL" "REAL" "REAL" "REAL" "INTEGER" "INTEGER" "INTEGER" "INTEGER" "REAL" "INTEGER"	The Random Sampling Procedures	30 28 42 45 43 54 79 67 54 52 69
HSTSET (A, N, L, W); HSTPUT (A, X); HSTPRINT (A, DEVOUT); HSTTRUE (A, M, V); HSTGROUP (A, M, V); HSTADD (A, B, C); HSTSUB (A, B, C); HSTIN (A); HSTOUT (A); HSTSAMP (A, U); HSTMULT (A, **K**, C);	- - - - - - - - - "INTEGER" -	The Histogram Procedures	43 87 196 40 129 14 14 183 32 92 75

Note: (1) If any of the Random Number or Sampling procedures are used then the

"INTEGER" "PROCEDURE" XBASERANDOM (P, Q);

must be included.

(2) If either of the procedures HSTADD or HSTSUB are used then the

"PROCEDURE" HST WORKS (A, B, C, ADD);

must be included.

(3) If the procedure HSTSAMP is used then the

"INTEGER" "PROCEDURE" RANDOM (A, B, U);

must be included.

(4) The following procedures must always be present.

```
"INTEGER" "PROCEDURE" STOREMAX;
"PROCEDURE" extract (first, first 1);
"PROCEDURE" insert (new, new 1);
"PROCEDURE" dopoint (i);
"PROCEDURE" close blocks;
"PROCEDURE" CALL (HAP, T);
"INTEGER" "PROCEDURE" NEXT;
"PROCEDURE" INITIALISE (A, B, C);
"PROCEDURE" PREPARE;
```

(5) The procedures STOREMAX, INITIALISE and XBASERANDOM are SIR coded blocks.

Chapter 12: THE GLOBAL VARIABLE 'TIME'

TIME is an integer global variable which holds a number representing the current time within the simulation. It is set to zero by the initialising procedure PREPARE and is adjusted as necessary by the procedure NEXT.

TIME is available to the user provided its value is not altered by the user's program.

Chapter 13: OPERATING INSTRUCTIONS

ESP900 together with the user's program is run as an ordinary 900 ALGOL program with SIR blocks.

ESP900 Tape 1 is a mnemonic ALGOL tape and contains all the ESP900 procedures. However, three procedures are SIR blocks and these are contained in SIR code form on ESP900 Tape 2.

The user is recommended to omit any procedures from the package that are not required.

Note the following details:

(a) The user's program must not have a title.

(b) The user's program must end with <u>two</u> additional "END"s.

(c) ESP maintains an

"INTEGER" "ARRAY" list [1 : max];

which holds details of the action calls. In general, the simulation program will run more efficiently if this list is made as large as possible.

At the start of the simulation run the message

ARRAY STORAGE AVAILABLE <I>
INPUT LIST SIZE

will be displayed on the on-line teleprinter. The user should subtract the space required by his own personal arrays from the integer I, and type the resultant integer terminated by a non-numeric character on the on-line teleprinter. This will cause the list to be set up to its maximum size for this particular simulation program.

If the user inputs 0 (zero), the array will be set up to a standard size of 500 locations.

<u>Note 1:</u> Allow 1 word per element for integer arrays and 2 words per element for real arrays. Also allow an extra 3 + 2d words for each array, where d is the number of dimensions of the array.

<u>Note 2:</u> If an on-line teleprinter does not exist then the message will be output on the punch and the reply must be input via the reader.

(d) If the program is compiled such that checking functions are compiled, then, in addition to any checks the user may have written in his program, each time the procedure NEXT is activated current time and the number of the next delayed action to be entered will be output on the current output device in the form

*TIME
*570
*NEXT
*6

The conditional actions are treated as NEXT 0 for this purpose. The facility may be used to aid 'debugging' simulation programs.

13.1 Redundant procedures

It is very unlikely that the user will wish to use all the simulation procedures on Tape 1 within his program. The following notes on the layout of Tape 1 may assist the user to edit out the unwanted procedures.

Tape 1 starts with the title

ESP900 PROVISIONAL;

followed by a list of essential declarations, (including procedures) which must not be omitted.

After the third halt code on the tape the random number procedures occur, ending with line "END" RANDOM;

The remaining procedures follow in order:

Fourth halt code
GEOMETRIC, LOGNORMAL, NEGEXP, NORMAL.
Halt code
FASTNORMAL, POISSON, BINOMIAL, PASCAL,
ERLANG, HYPERGEOM.
Halt code
HSTSET
Halt code
HSTPUT, HSTTRUE, HSTPRINT.
Halt code
HSTGROUP.
Halt code
HSTWORKS, HSTADD, HSTSUB, HSTMULT.
Halt code

HSTIN, HSTOUT
Halt code
HSTSAMP

All the procedures listed above end with a line of the form

"END" NAME;

where NAME is the procedure name.

Example

In order to produce a tape which has the title SIM2 and contains everything except the procedures BINOMIAL to ERLANG and HSTTRUE to HSTSUB the following edit could be used:

```
DC;
IL SIM2;
FL "END"POISSON;
DL "END"ERLANG;
FL "END" HSTPUT;
DL "END"HSTSUB;
FL "END"HSTSAMP
SH
IH
(H)
```

The users simulation program could be edited onto the same tape by following the SH with another SH or other suitable command.

Chapter 14: ERROR MESSAGES

During the course of a simulation program errors may occur in the use of ESP, in which case, a message will be displayed on the on-line teleprinter and the program will stop. It is not possible to continue after an error. If an on-line teleprinter is not fitted then the error messages are output on the punch.

(a) ERR 1

The array list 1: max is too small for the simulation to continue. Should the user wish to run the simulation further, the program size must be reduced to make room for a larger list.

(b) ERR 2

There are no more action calls to be obeyed. The simulation is presumed complete.

(c) ERR 3

A call of procedure CALL contains an illegal parameter.

Either (i) HAP is less than 1.

or (ii) T is less than 0.

or (iii) the simulation has lasted more than 131070 simulated time units.

or (iv) the number of parameters (set in SEND [0]) is less than 0 or greater than 10.

(d) ERR 4

Histograms used by either HSTADD or HSTSUB are incompatible.

(e) ERR 5

Histograms used by HSTMULT are either incompatible or K, the multiplying constant, is negative.

(f) ERR 6

Error during input of an histogram by HSTIN. The calculated number of observations does not agree with the non-zero value on the input tape.

Chapter 15: RESTRICTION

ESP imposes the restriction that the simulation experiment may not last longer than 131071 simulated time units.