Chapter 6
Simulation Languages

Introduction

A computer **simulation language** describes the operation of a simulation on a computer. There are two major types of simulation: continuous and discrete event though more modern languages can handle combinations. Most languages also have a graphical interface and at least simple statistical gathering capability for the analysis of the results. In a sense, these languages are similar to language like Fortran, java or vb.net but also include specific features to facilitate the modelling process. Some example of modern simulation languages are CPSS/H, GPSS, SLX, SIMSCRIPT, DSL etc.

Merits of Simulation Language

- Since most the features to be programmed are in-built simulation language like comparatively less programming time and effort.
- Since the simulation language consists blocks, specially constructed to simulate the common features, they provide a natural framework for simulation modeling.
- The simulation models coded in simulation languages can easily be changed and modified.
- The error detection and analysis is done automatically is simulation language.
- The simulation models developed in simulation languages, especially the specific application packages, called simulators, are very easy to use.
- Random number generator and related set of tools
- Provides tools for reporting result.
- Simulation clock or mechanism for advancing simulated time.

Comparisons / Advantage of using simulation packages over programming language

1. Simulation package automatically provide features needed to build a simulation model, resulting significant decrease in programming time and project cost.
2. It provides natural framework for simulation modelling.
3. Simulation models are generally easier to modify and maintain when written in a simulation package.
4. They provide better error detection because many potential types of error are checked for automatically.

However, Simulation model are still written in programming language because it offers following advantages.

1. Analysis modeler already know a programming language this is often not the case with the simulation package.
2. Simulation model efficiently written in C or C++ may require less execution time then a model developed by simulation package.
3. A simulation package is designed to address a wide variety of systems with one set of modelling constructs however, a C program may be more closely tailored to a particular application.
4. Programming language may sometime allow greater programming than certain simulation package.

Types of Simulation Language

. There are following types of simulation languages:

1. Continuous system simulation language (CSSLS)
2. Discrete system simulation language (DSSLS)
3. Hybrid system simulation language (HSL)

Continuous System simulation language

Before digital computers come into common use, analog computers were being used for simulating continuous system. The system in an analog computer was represented by the block of electronic devices such as operational amplifier to carry out the required operation. As soon as the digital computer came, it provides many advantages over analog computer like greater accuracy, freedom for scaling, easily stored for reuse etc. Therefore, a special program package were implemented on digital computer to make a digital computer appear like analog computer. This is called a digital analog simulators. Example: DEPI, DEPI-4, DAS, MIDAS, IBM 1130, CSMP etc

.

Types:

1. Block structured continuous simulation language
   These are designed to simulate only those system that could be represented as analog block diagrams and thus their application are is limited. It requires user to prepare a analog computer type block diagram and then input this diagram.

2. Expression based languge
   A block oriented continuous system simulation languge is limited because of its small specialized vocabulary. Since, the system must be represented first in analog block diagram so this languge are suited only for analog system and not for those who model the system as set of equations. Expresssion based language or statement based language overcome this drawback. Expression based language do not follow block structure diagram but directly implements the differential equation of the model.

Discrete System simulation language

These languages are used to simulate discrete system and provide the following facilities.

- Automatic generation of random no.
- Automatic data collection
- Statistical analyses of data
- Reporter generators, etc.

The other classification of discrete simulation languages is based on the general world view inherent in the language. Event, activity and process form the basis of three primary conceptual frameworks (world view) within discrete event simulation.

Types

1. Event oriented language
   In an event oriented language each event is represented by an instantaneous occurrence in simulated time and must be scheduled to occur in advance when a proper set of conditions exists. The system image changes i.e. state of system changes at the occurrence of event. Example, SIMSCRIPT, GASP
2. Activity oriented language
   In an activity oriented language, the discrete occurrences are not scheduled in advance. They are created by a program which contains description of the conditions under which an activity can take place. The activity oriented language have two components: Test component and action component. Here the event occurrence must be controlled by a cyclic scanning activity program. Example, MILITRAN
3. Process oriented languages
   Here a single process routine, composed of number of segments describe a sequence of activities. Each segment behaves has an independently controlled program. On receiving control, only the statement composing the segments are executed, and then control is returned. Example, ASPOL, SIMUFOR
4. Transaction flow oriented languages.
   These languages are the sub category of process oriented language where flow of activities passes through specially defined block System model is represented by a flow chart consisting of language. The program creates transaction(entities), executes them in the blocks and moves them along the flowchart. These languages are flow chart oriented; the best known example is GPPS.

Hybrid System Simulation Language

Some language has been developed which are suitable for both discrete as well as continuous models. This type of language is called hybrid system simulation language or combined simulation language. These are written particularly for system models in which some of the variable changes continuously and other variable changes discretely. Example, GASP IV

GPSS (General Purpose Simulation System)

- GPPS, one of the earliest DSL was developed by Geoffrey Gordon (1961- 1962). The first release of this language was implemented on the IBM 704, 709 and 7090 computers. Later on improved and more powerful versions have been developed and implemented, including GPSS 2^{ND} , GPSS 3^{RD} (1965), GPSS 1360 (1967) and GPSS V (The latest version).
- GPSS was designed especially for those analyses who weren't necessary computer programmer because they do not write programmed in the logic as the SIMCRIPT programmer does. Instead he constructs a block diagram – an n/w of inter connected blocks, each performing a special simulation oriented function. GPSS is particularly suited for traffic and queuing system.

Characteristics of GPSS

- Designed for analyst not programmers
- GPSS is described as block diagram in which **the block represents the activities**, and the lines joining the block indicates the sequence in which the activities can be executed.
- GPSS V is a set of 48 different block types which perform some specific task.
- Machine efficiency is poor because GPSS is interprets system.
- Restricted to simple queuing problems.

GPSS Block Diagram

The development of a simulation model in GPSS is a block-by-block construction. A set of standard block is arranged in the form of a block diagram that represents the flow of entities(transaction) through the various paths of the system. Each block represents a step in the action of the system and links, joining the blocks, represent the sequence of events that can occur.

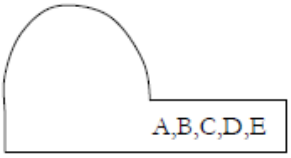
To build a block diagram, it is essential to have a completed description of the system. The meanings of the blocks used in the system must be clearly defined. Each block must be assigned the block time, i.e. the time which the execution of the block will take.

- A set of block types have been designed, which can be used in the construction of a block diagram. Each block type can be used any number of times in a block diagram, but the total number of blocks should not exceed 2047. On the completion of the block diagram, each block is assigned a number, between 1 and 2047, automatically called the block number or block identification number or location.
- The system to be simulated in GPSS is described as a block diagram in which block represents activities and lines joining the block indicate the sequence in which the activity can be executed.
- Each block performs a simulated oriented function.
- GPSS V provides a set of 48 different blocks, each of which can be used repeatedly. Each block has a name and specific task to perform. Each block type has a no. of data field such as A, B, C, and so on.
- The entities of the system being simulated are called as transaction. Eg; costumer in a queuing system, message in communication system.

Typical blocks are;

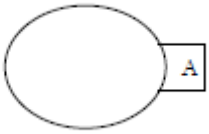
1. GENERATE: Create transaction and place on future event chain

- GENERATE A,B,C,D,E
  - create Xact for future entry into system
  - birth block
    - A: mean value of interarrival time
    - B: half-range of interarrival time
      - $B \leq A$
    - C: start delay time
    - D: creation limit
    - E: priority
- interarrival time uniformly distributed in A±B



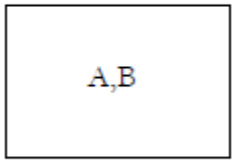
2. TERMINATE: Removes transaction form the system.

   TERMINATE A
   - Destroys an Xact
   - Death block
   - A: Termination count increment
   - when Xact move into block , transaction count(tc) value is incremented by A and continues until it reach some predefined simulation steps.



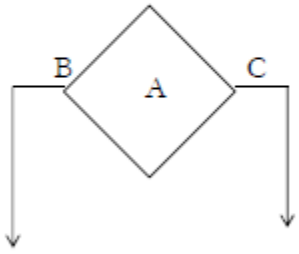
3. Advance: The block type 'ADVANCED' is concerned with representing the expenditure of time. The program computes an interval of time called an action time for each transaction as it enters an 'ADVANCED' blocks and transaction remain at this block up to that action time.

- ADVANCE A,B
  - delay movement
    - A: mean value for delay amount
    - B: half range for delay amount
  - delay amount uniformly distributed in A±B



4. Transfer: The 'TRANSFER' block allows some location other than the next sequential location to be selected i.e. when there is choice of activities, we use this block by stating the condition for the choice. The choice is normally between 2 blocks refer to as next block 'A' and 'B'.

- TRANSFER A,B,C
  - jump to new location
  - A: probability or mode
    - determines which block to choose
    - probability shows transfer to location C
    - if empty: unconditional transfer to location B
    - Both: chooses empty block (B first if both empty)
  - B: block location
  - C: block location



Note: A is selection factor, B and C fields are exit1 and exit2 respectively.
Example: TRANSFER 0.1, ACC, REJ means 10% go to location REJ(exit2) and 90% to ACC(exit1)

Characteristics of the Blocks

1. Transactions(X_{acts})

In each system represented by a block diagram, some entities pass through the system. In petrol pump system, they may be vehicles, in a production system they may be parts and in supermarket system they may be costumer. Those entities are referred to as Transactions. In simulation the transaction are created, which move through the block diagram in the same way, as the entities pass through the actual system being simulated. There can be many transactions simultaneously moving through the block diagram.

2. Block / Action Time:

The block time also called action time is an integer giving the number of units required to execute the action represented by the block. The program computes an action time for each transaction entering a block to represent the time taken by the system action simulated by the block. The program compute and interval of time called action time for each transaction as it enters and ADVANCE block, and the transaction remains at the block for this interval of simulated time before attempting to proceed. The action time may be a fixed interval or a random variable. An action time is defined by giving a mean and modifier. Example, T=A±B where A is the mean and B is the modifier.

3. Succession of Events

The program maintains records of when each transaction in the system is due to move. It proceeds by completing all movements that are scheduled for execution at a particular instant of time and that can be logically performed. When there are more than one transaction due to move, the program processes transactions in the order of priority or FCFS basis. Once the program begun moving a transaction, it continuous to move the transaction through the block diagram until the transaction enters ADVANCE(wake up other transaction and return to itself when action time is expended) or TERMINATE (transaction is removed from simulation) block or the transaction is blocked(action the transaction is attempting to perform can't be perform currently).

4. Selection Factor/Choice of path

The TRANSFER block allow some location other than the next sequential location to be selected. The choice is normally between two blocks referred to as blocks A and B (also called as exit1 and exit 2). The choice of which exits to be followed is given by a number called the selection factors, S ($0<S<1$) which is entered in the block. S is the probability of selecting exit 2 and $1 - S$ is the probability of selecting exit 1. Also if the selection facto is BOTH, it first chooses empty block A i.e. exit1

5. Items of Equipment

In each system, there are physical equipment, which perform some operation on the transactions. Machine tools in a production shop perform machining operations on work pieces. Example, in transportation system, a toll booth on a road is equipment, while vehicles are transactions. In Factory Manufacturing system, an inspector is equipment, while parts are transaction, etc. The item of equipment may operate upon transactions individually or may handle groups.

6. Storage and Facilities

Item of equipment can be classified into storage and facilities, depending upon the capacity for handling the transactions. An item of equipment, which can handle only one transaction at a time, is called a facility while the items of equipment, which can handle a large number of transactions at the same time is called store.

Example: In communication system, message is transaction, switch(pass only one message) is facility while trunk(collection of wire carrying several messages simultaneously) is storage.

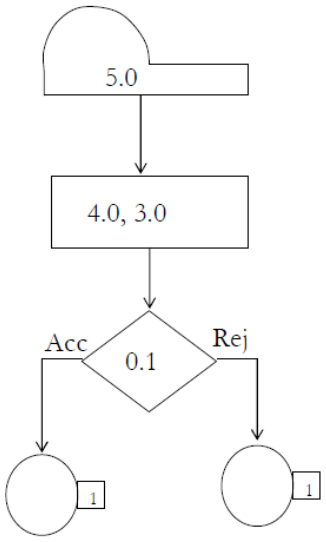
GPSS PROGRAM Example

Example 1: Simulation of manufacturing shop

In a manufacturing shop, a machine tools turns out parts at the rate of one every five minutes. As they are finished the posts go to inspector, who takes 4±3 minutes to examine each part and rejects 10% of parts. Each part will be represented by one transaction and time unit selected for problem will be one minute. Simulate for 100 parts to leave the system.

A GPSS block diagram for this system is as follows:

4-3
4+3
1,2,3,4,5,6,7



GPSS coding of the above block diagram of the manufacturing shop:

```
GERNERATE          5
ADVANCE            4, 3
TRANSFER           0.1, ACC, REJ
ACC    TERMINATE        1
REJ    TERMINATE        1
START              1000
```

Figure Description;
A GENERATE block is used to represent the output of machine by creating one transaction every five minutes of time. The ADVANCE blocks with a mean of four and modifier of three is used to represent inspection will therefore be anyone of values 1,2,3,4,5,6,7. After completion of the inspection transaction go to or a TRANSFER block with a selection factor of 0.1, so that 90% of the parts go to next location i.e. exit1 call ACC and 10% go to another location i.e. exit2 called REJ. Since there is no further interest both location reached from the transfer blocks are terminate blocks. The program runs until a certain **count** is reached as a result of transactions terminating. Field A of terminate block carries a number indicating by how much the termination count should be incremented when transaction terminates at that block. The control statement START has a field A that indicates a value the terminating counter should reach to end the simulation. When the simulation is completed, the program automatically generate a report, in a prearranged format.

Note:

i. In this case exit1(i.e.ACC) is the next sequential block so it is allowed to omit the name ACC in both the Transfer Field B and the location field of first terminate block. The GPSS code becomes like this TRANSFER 0.1, , REJ. Both comma must be given to indicate B is missing.
ii. If transfer block is used as unconditional mode, then we need not have to specify field A. however comma must be given to indicate A is missing. The GPSS code become like this TRANSFER , REJ

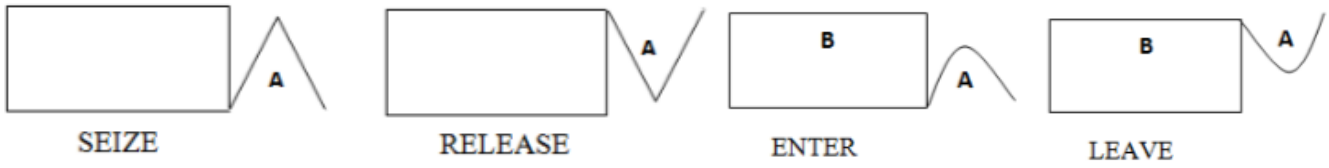
Facilities and storage;

A facility is defined as an entity or resources that can be engaged by a single transaction at a time. Storage is defined as an entity or resource that can be occupied by much transaction at a time up to some pre-determined limit.
Once a transaction seizes (holds) a facility any other transaction trying to seize the same facility is delayed until the first transaction releases the facility (resources) but not in case of storage until storage limit reaches. Some example of the system entities might be interpreted in different system are

Types of system	Transaction	Facility	Storage
Communication	Message	Switch	Trunk
Transportation	Car	Toll both	Road
Data processing	Record	Key stroke	Computer memory
Computer	Process	CPU	memory

There are additional four block types concerned with using facilities and storages

1. SEIZE: The SEIZE block allow a transaction to engage a facility if it is available.
2. RELEASE: The RELEASE block allows the transaction to disengage the facility
3. ENTER: The ENTER block allows a transaction to occupy space in storage, if it is available
4. LEAVE: The LEAVE block allows a transaction to give up the occupied space.



Capture facility A | Release Facility A | Capture B unit of Storage A | Release B unit of Storage A

Field A in each case indicates which facility or storage in intended, and the choice is usually marked in flag attached to the symbols of the block. If the field B in ENTER and LEAVE blocks are blank, the storage contents are changed by 1. If there is a numeric (≥1), then the content changes by that value.

Example:
Let us consider a situation as below;

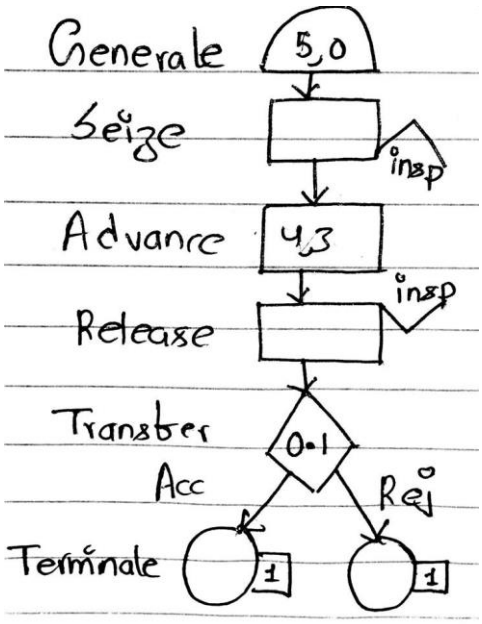
SEIZE	CPU
ADVANCE	7
RELEASE	CPU

Here CPU is a facility and it is seems that a transaction needs to use the resource for 7 times units. Any other transaction arriving at the block 'SEIZE' is refused to enter until the former transaction has entered RELEASE block.

Resources which can be shared by several transactions are modeled using storage.
Suppose we want to model a computer system which has 64kb of memory then we might declare 'MEMORY STORAGE 64' and then a request for 16kb memory might be represented by the sequences of blocks.
ENTER MEMORY 16
LEAVE MEMORY 16
As the facilities, a transaction arriving at ENTER block at a time when it is used by other transactions, is delayed until the previous transaction release the necessary memory with. A transaction controlling and facility can be interrupted or preempted by other transactions.

Example
In a manufacturing shop, a machine tools turns out parts at the rate of one every five minutes. As they are finished the parts go to inspector, who takes 4±3 minutes to examine each part and rejects 10% of parts. Each part will be represented by one transaction and time unit selected for problem will be one minute. Simulate for 100 parts to leave the system assuming that there is only one inspector.
Note: since there is only one inspector, it is necessary to represent the inspector by a **facility**, to simulate the fact that only one part at a time can be inspected.

A GPSS block diagram for this system is as follows:



GPSS coding of the above block diagram of the manufacturing shop:

GENERATE 5	Create parts
SEIZE insp	Get Inspector
ADVANCE 4,3	Inspect
RELEASE insp	Free Inspector
TRANSFER 0.1,Acc,Rej	Select reject
Acc TERMINATE 1	Accepted Parts
Rej TERMINATE 1	Rejected Parts
START 100	Run 100 parts

Example(*top*)
In a manufacturing shop, a machine tools turns out parts at the rate of one every five minutes. As they are finished the posts go to inspector, who takes 4±3 minutes to examine each part and rejects 10% of parts. Each part will be represented by one transaction and time unit selected for problem will be one minute. Simulate for 100 parts to leave the system assuming that there are three inspectors
Note: if more than one inspector is available, they can be represented as a group by **Storage** with a capacity equal to the number of inspectors.

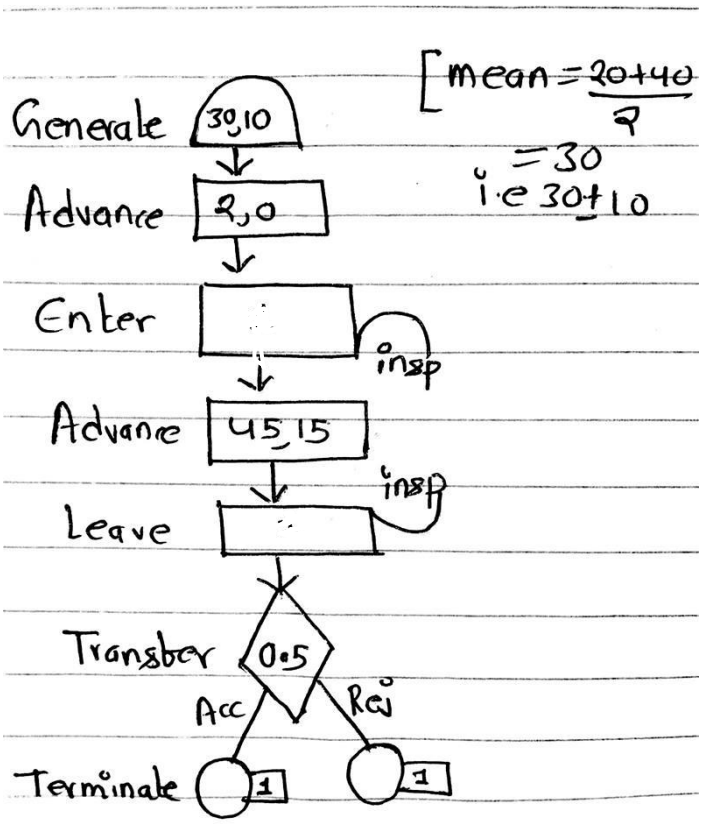
GPSS code:

	GENERATE 5	create parts
	ENTER insp	get an inspector
	ADVANCE 4,3	inspect
	LEAVE insp	free inspector
	TRANSFER 0.1,Acc,Rej	select reject
	Acc TERMINATE 1	accepted part
	Rej TERMINATE 1	rejected part
insp	STORAGE 3	Numbers of inspector
	START 100	Run 100 parts

Note: STORAGE is a control statement which assigns the capacity of storage.

Consider a factory that manufactures football taking 20 to 40 minutes. The ball is moved from the generation to the inspection machine taking a 2 minutes. There are 3 inspection machine at one place and need 30 to 60 minutes for inspection and reject 30% of football. Simulate for 1000 transaction Draw GPSS block diagram to simulate this system.

GPSS Block Diagram

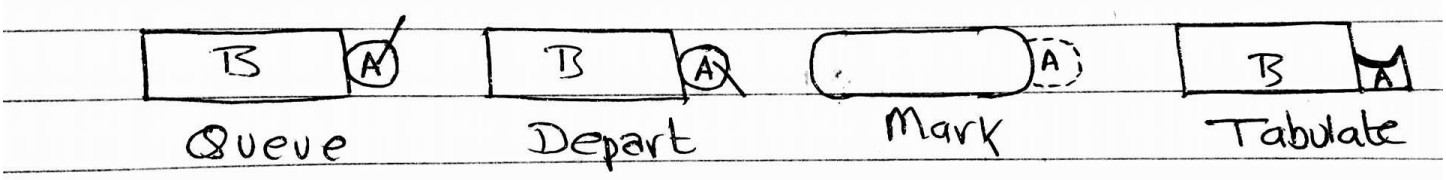


GPSS code

```
        GENERATE 30,10
        ADVANCE 2, 0
        ENTER insp
        ADVANCE 45, 15
        LEAVE insp
        TRANSFER 0.5, Acc, Rej
        Acc TERMINATE 1
        Rej TERMINATE 1
insp    STORAGE 3
        START 1000
```

Gathering Statistics

Block types such as QUEUE, DEPART, MARK and TABULATE are used to gather statistic about the system performance but not to represent the system action.



When the condition for advancing a transaction is not satisfied several transactions may be kept waiting at a block and due to this the QUEUE block increase and DEPART block decreases the number in field A i.e. queue block increases the

queue number while depart block decreases the queue number. A is queue number, B is the quantity by which the queue number is being increased

The MARK and TABULATE blocks are used to measure the length of time taken by transaction to move the system or parts of the system. The MARK block simply notes the **arrival time** on the transaction and the TABULATE blocks subtracts the time noted by MARKED block from the time of arrival at the TABULATE block.

Example

Consider the example of manufacturing shop with 3 inspectors. Parts will now accumulate on the inspectors works bench if inspection doesn't finish quickly enough. Simulate the situation for 1000 parts to measure how long the parts take to be inspected, excluding their waiting time in queue.

Note:

i. Up to the above examples, the properties of generate block is such that if a transaction is unable to leave a block at the time it is created, no further transactions are generated until the block is cleared so if all the inspectors are busy, the machining of further parts stops until the machine is cleared.

ii. So if all inspectors are busy then queue is used to accumulate incoming parts/ transactions.

GPSS block Diagram:

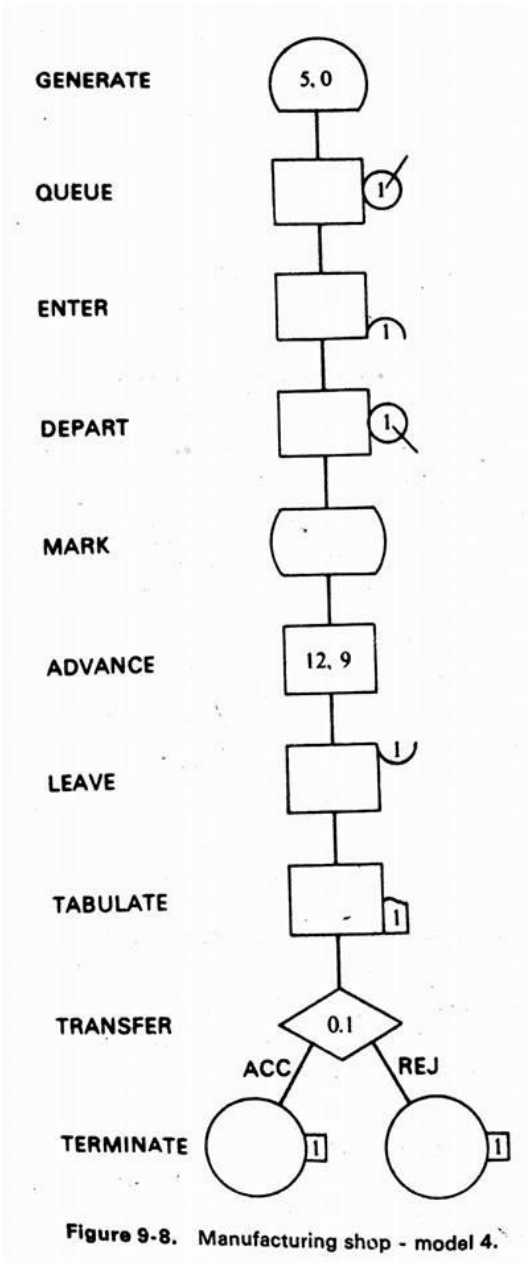


Figure 9-8. Manufacturing shop - model 4.

GPSS Code

- In the above solution, A QUEUE block using queue number 1 is placed immediately before ENTER block and a DEPART block is placed immediately after the ENTER block to remove the part from the queue when inspection begins.
- If some inspectors are free for inspection, then transaction doesn't have to wait in queue otherwise they have to wait. The program will automatically measure the length of stay in queue.
- The MARK and TABULATE block will measure how long the parts take to be inspected, excluding their waiting time in the queue.
- If MARK block is omitted , the tabulated time is the time since the transaction first entered the system.

```
          GENERATE    5              CREATE PARTS
          QUEUE       1              QUEUE FOR AN INSPECTOR
          ENTER       1              GET AN INSPECTOR
          DEPART      1              LEAVE QUEUE
          MARK
          ADVANCE     12,9           INSPECT
          LEAVE       1              FREE INSPECTOR
          TABULATE    1              MEASURE TRANSIT TIME
          TRANSFER    .1,ACC,REJ     SELECT REJECTS
ACC       TERMINATE   1              ACCEPTED PARTS
REJ       TERMINATE   1              REJECTED PARTS
*
1         STORAGE     3              NUMBER OF INSPECTORS
1         START       1000           RUN FOR 1000 PARTS
*
```

Example
Simulate one day of operation of a barber shop for 200 customers. Customers arrive in a barber shop at the rate of 18±6 minutes , enter the shop, queue if the barber is busy, get their hair cut which take 16±4 minutes on a first-come first-served basis, and then leave the shop.

GPSS code

```
GENERATE 18,6       Customer arrive every 18±6 m
QUEUE    Chairs     Enter the line
SEIZE    hari       Capture the barber
DEPART   Chairs     Away from the line
ADVANCE  16,4       Get a hair cut in 16±4 mn
RELEASE  hari       Free the barber
TERMINATE 1         Leave the shop

START    200
```

Example
Workers come to a supply at the rate of one every 5±2 minutes. Their requisitions are processed by one of two clerks who take 8±4 minutes for each requisition. The requisition are then passed to a single storekeeper who fills them one at a time, taking 4±3 minute for each request. Simulate queue of workers and measure the distribution of time taken for 1000 requisitions to be filled.

```
************************************************************
* GPSS Simulation: Supply Requisition Processing          *
* Simulates the process of workers' requisitions          *
************************************************************

SIMULATE 1000

* Define entities and parameters
GENERATE 5,2        * Workers arrive every 5 ± 2 minutes
QUEUE ClerkQ        * Queue for clerks
SEIZE Clerk         * Requisition is processed by a clerk
DEPART ClerkQ       * Leave the clerk queue
ADVANCE 8,4         * Clerks take 8 ± 4 minutes to process requisitions
RELEASE Clerk       * Release the clerk

* Pass to storekeeper
QUEUE StorekeeperQ  * Queue for the storekeeper
SEIZE Storekeeper   * Storekeeper processes requisition
DEPART StorekeeperQ * Leave the storekeeper queue
ADVANCE 4,3         * Storekeeper takes 4 ± 3 minutes to fill requisition
RELEASE Storekeeper * Release the storekeeper

* Termination condition
TERMINATE 1         * Each requisition completed contributes to the termination count

* Define facilities
FACILITY Clerk,2    * Two clerks are available
FACILITY Storekeeper

* End program
END
```

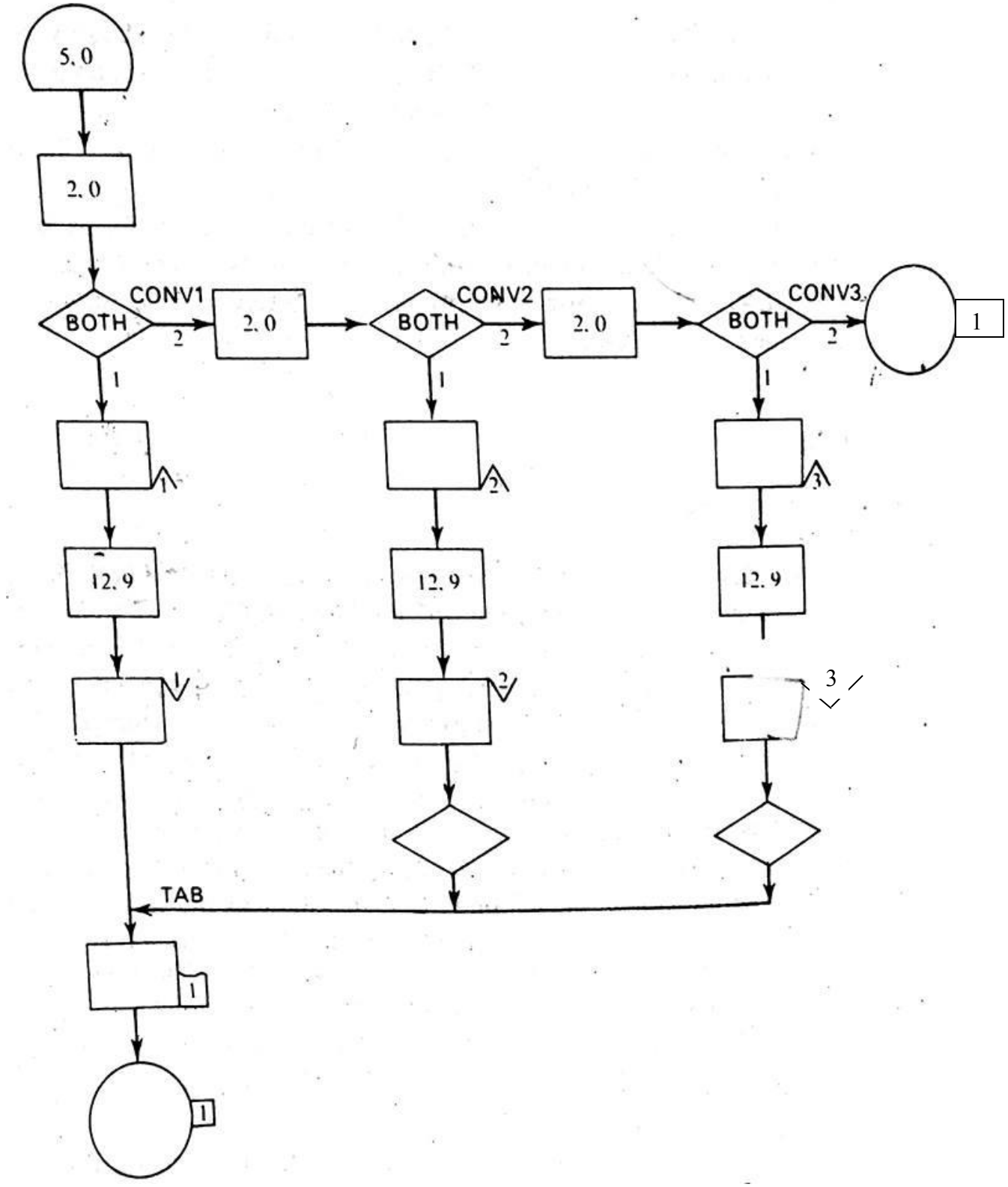
Workers come to a supply at the rate of one every 5±2 minutes. Their requisitions are processed by two clerks who take 8±4 minutes for each requisition. The requisition are then passed to a single storekeeper who fills them one at a time, taking 4±3 minute for each request. Simulate queue of workers and measure the distribution of time taken for 1000 requisitions to be filled. generate gpss code for above statement.

```
************************************************************
* GPSS Simulation: Supply Requisition Processing          *
* Simulates the process of workers' requisitions          *
************************************************************

SIMULATE 1000

* Define entities and parameters
GENERATE 5,2        * Workers arrive every 5 ± 2 minutes
QUEUE ClerkQ        * Queue for clerks
ENTER ClerkStorage,1 * Workers enter clerk storage (shared by two clerks)
DEPART ClerkQ       * Leave the clerk queue
ADVANCE 8,4         * Clerks take 8 ± 4 minutes to process requisitions
LEAVE ClerkStorage,1 * Workers leave clerk storage

* Pass to storekeeper
QUEUE StorekeeperQ  * Queue for the storekeeper
SEIZE Storekeeper   * Storekeeper processes requisition
DEPART StorekeeperQ * Leave the storekeeper queue
ADVANCE 4,3         * Storekeeper takes 4 ± 3 minutes to fill requisition
RELEASE Storekeeper * Release the storekeeper

* Termination condition
TERMINATE 1         * Each requisition completed contributes to the termination count

* Define storage and facilities
STORAGE ClerkStorage,2 * Two clerks are represented by storage
FACILITY Storekeeper

* End program
END
```

```
************************************************************
* GPSS Simulation: Supply Requisition Processing           *
* Simulates the process of workers' requisitions           *
************************************************************

SIMULATE 1000

* Define entities and parameters
GENERATE 5,2         * Workers arrive every 5 ± 2 minutes
ENTER ClerkStorage,1 * Worker enters clerk storage (shared by two clerks)
ADVANCE 8,4          * Clerk processes the requisition (8 ± 4 minutes)
LEAVE ClerkStorage,1 * Worker leaves clerk storage

* Pass to storekeeper
SEIZE Storekeeper    * Storekeeper begins processing requisition
ADVANCE 4,3          * Storekeeper fills the requisition (4 ± 3 minutes)
RELEASE Storekeeper  * Storekeeper releases the requisition

* Termination condition
TERMINATE 1          * Each requisition completed contributes to the termination count

* Define storage and facilities
STORAGE ClerkStorage,2  * Two clerks are represented by storage
FACILITY Storekeeper

* End program
END
```

Condition Transfer

The TRANSFER block can be used for both conditional and unconditional transfer mode. Consider again the case of three inspectors but suppose that the manufactured parts are put on a conveyor, which carries the parts to the inspector. It takes 2 minutes for a part to reach the first inspector who will take the part if he is not busy otherwise parts are passed to second inspector which takes further 2 minutes to reach second inspector. If he is also busy then parts are passed to third inspector which take another 2 minute to reach third inspector, otherwise they are lost.

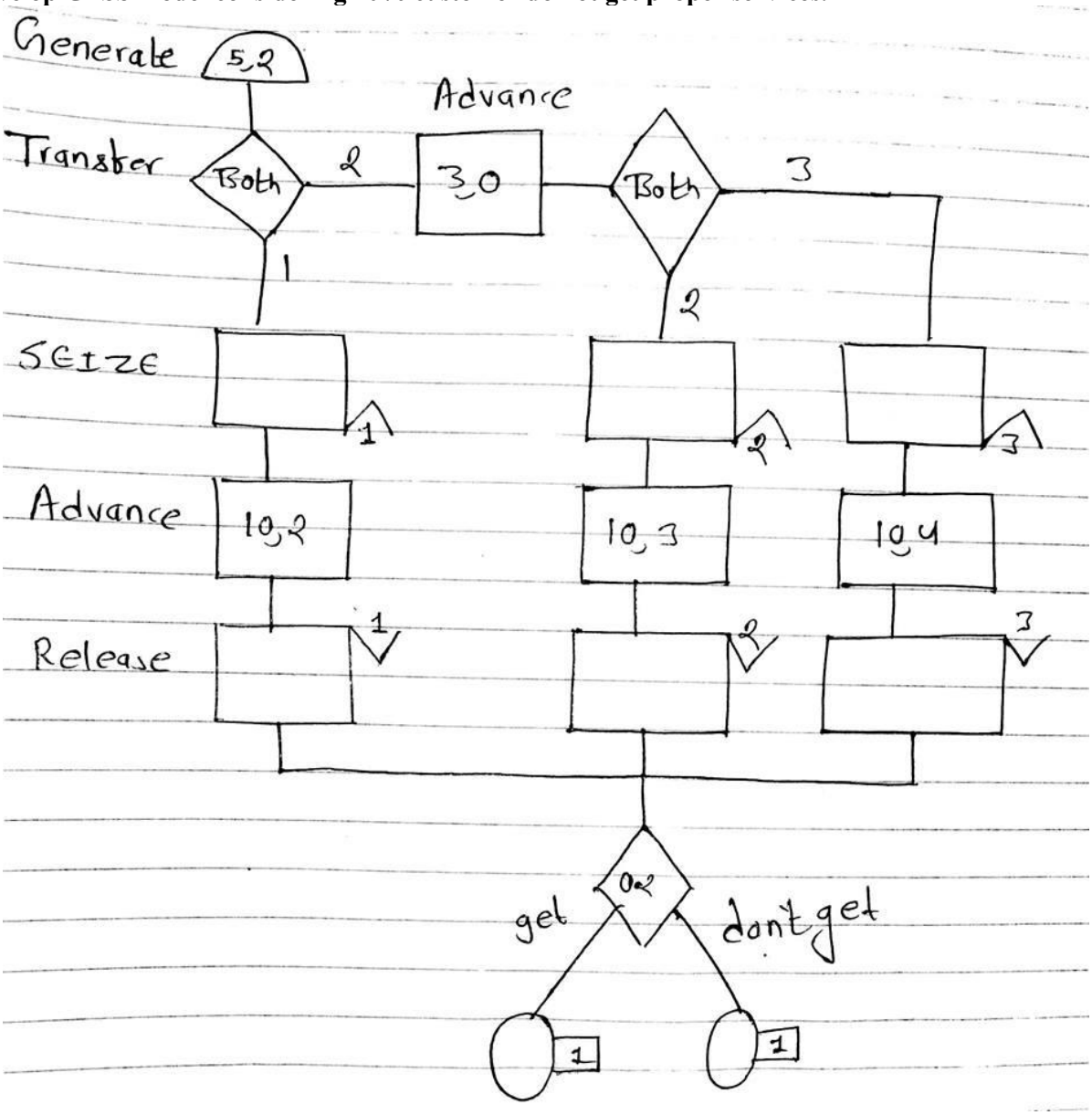


i. In the above diagram, the movement of conveyer belt is represented by ADVANCE block.

ii. As a transaction leave the advance block, it test whether an inspector is available by entering a TRANSFER block with a selection factor set to BOTH.
iii. In the TRANSFER block the selection mode BOTH implies B first if both empty.
iv. TRANSFER block lead to SEIZE block that represent 1 inspector here 1, 2 and 3.(NOTE : we also can give name like ins1, ins2, ins3)
v. When parts finish inspection, the transaction goes to single a single TABULATE block where the transit time is recorded.
vi. Because of the rule by which transactions normally pass from one block to next higher numbered block, only one of the three release block that complete the inspection is able to pass transactions directly to the TABULATE block.
vii. The other pass the transaction to TABULTE block using unconditional TRANSFER Block.

Example

Consider a bank with 3 service counter where customer arrival time is in average of 5, with a variance of 2 minutes. If any customers find the first service counter busy, he/she goes to another service counter but it takes 3 extra minutes to move into the another service counter, similar condition for reaching to third counter. It takes average of 10 minutes to provide service to any customer with 2, 3, 4 minutes' variance respectively at counter 1,2 and 3. Develop GPSS model considering 20% customer do not get proper services.



GPSS BANK CODE

```
          GENERATE 5,2
          TRANSFER BOTH,,CONV1
          SEIZE 1
          ADVANCE 10,2
          RELEASE 1
      TAB TRANSFER 0.2 ACC REJ
      ACC TERMINATE 1
      REJ TERMINATE 1
CONV1     ADVANCE 3,0
          TRANSFER BOTH,,CONV2
          SEIZE 2
          ADVANCE 10,3
          RELEASE 2
          TRANSFER ,TAB
CONV2     ADVANCE 3,0
          SEIZE 3
          ADVANCE 10,4
          RELEASE 3
          TRANSFER ,TAB

          START 1000
```

GPSS World Simulation Report - BANK.6.1

Monday, January 20, 2025 07:18:58

START TIME	END TIME	BLOCKS	FACILITIES	STORAGES
0.000	4979.807	19	3	0

NAME	VALUE
ACC	7.000
CONV1	9.000
CONV2	15.000
REJ	8.000
TAB	6.000

LABEL	LOC	BLOCK TYPE	ENTRY COUNT	CURRENT COUNT	RETRY
	1	GENERATE	1002	0	0
	2	TRANSFER	1002	0	0
	3	SEIZE	401	0	0
	4	ADVANCE	401	1	0
	5	RELEASE	400	0	0
TAB	6	TRANSFER	1000	0	0
ACC	7	TERMINATE	826	0	0
REJ	8	TERMINATE	174	0	0
CONV1	9	ADVANCE	601	1	0
	10	TRANSFER	600	0	0
	11	SEIZE	350	0	0
	12	ADVANCE	350	0	0
	13	RELEASE	350	0	0
	14	TRANSFER	350	0	0
CONV2	15	ADVANCE	250	0	0
	16	SEIZE	250	0	0
	17	ADVANCE	250	0	0
	18	RELEASE	250	0	0
	19	TRANSFER	250	0	0

FACILITY	ENTRIES	UTIL.	AVE. TIME	AVAIL.	OWNER	PEND	INTER	RETRY	DELAY
1	401	0.802	9.959	1	1001	0	0	0	0
2	350	0.700	9.954	1	0	0	0	0	0
3	250	0.493	9.829	1	0	0	0	0	0

FEC XN	PRI	BDT	ASSEM	CURRENT	NEXT	PARAMETER	VALUE
1002	0	4979.951	1002	9	10		
1003	0	4980.149	1003	0	1		
1001	0	4983.143	1001	4	5		

```
                    * MANUFACTURING SHOP - MODEL 5
*
                GENERATE    5           CREATE PARTS
1               ADVANCE     2           PLACE ON CONVEYOR
2               TRANSFER    BOTH,,CONV1 MOVE TO FIRST INSPECTOR
3               SEIZE       1           GET FIRST INSPECTOR
4               ADVANCE     12,9        INSPECT
5               RELEASE     1           FREE INSPECTOR
6         TAB   TABULATE    1           MEASURE TRANSIT TIME
7               TERMINATE   1
8
*
        CONV1   ADVANCE     2           PLACE ON CONVEYOR
9               TRANSFER    BOTH,,CONV2 MOVE TO SECOND INSPECTOR
10              SEIZE       2           GET SECOND INSPECTOR
11              ADVANCE     12,9        INSPECT
12              RELEASE     2           FREE INSPECTOR
13              TRANSFER    ,TAB
14
*
        CONV2   ADVANCE     2           PLACE ON CONVEYOR
15              TRANSFER    BOTH,,CONV2 MOVE TO THIRD INSPECTOR
16              SEIZE       3           GET THIRD INSPECTOR
17              ADVANCE     12,9        INSPECT
18              RELEASE     3           FREE INSPECTOR
19              TRANSFER    ,TAB
20
*
21      CONV3   TERMINATE
*
        1       TABLE       M1,5,5,10   TABULATION INTERVALS
*
                START       10,NP       INITIALIZE WITH TEN PARTS
                RESET
                START       1000        MAIN RUN
```

RELATIVE CLOCK 5588 ABSOLUTE CLOCK 5660

BLOCK COUNTS

BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL	BLOCK	CURRENT	TOTAL
1	0	1117	11	0	343	21	0	118
2	0	1117	12	0	343			
3	0	1118	13	0	344			
4	0	406	14	0	344			
5	1	406	15	1	369			
6	0	406	16	0	368			
7	0	1000	17	0	250			
8	0	1000	18	0	250			
9	0	712	19	0	250			
10	0	712	20	0	250			

****
* FACILITIES *
****

FACILITY	NUMBER ENTRIES	AVERAGE TIME/TRAN	-AVERAGE UTILIZATION DURING- TOTAL TIME	AVAIL. TIME	UNAVAIL. TIME	CURRENT STATUS	PERCENT AVAILABILITY
1	407	11.885	.865				100.0
2	344	11.956	.736				100.0
3	250	12.244	.547				100.0

****
* TABLES *
****

TABLE 1
ENTRIES IN TABLE 1000 MEAN ARGUMENT 15.693 STANDARD DEVIATION 5.757 SUM OF ARGUMENTS 15694.000

UPPER LIMIT	OBSERVED FREQUENCY	PER CENT OF TOTAL	CUMULATIVE PERCENTAGE	CUMULATIVE REMAINDER	MULTIPLE OF MEAN	DEVIATION FROM MEAN
5	27	2.69	2.6	97.2	.318	-1.857
10	202	20.19	22.8	77.1	.637	-.988
15	269	26.89	49.7	50.2	.955	-.120
20	248	24.79	74.5	25.4	1.274	.747
25	228	22.79	97.3	2.6	1.592	1.616
30	26	2.59	100.0	.0	1.911	2.484

REMAINING FREQUENCIES ARE ALL ZERO

Figure 9-11. Coding and results for model 5.

PROGRAM CONTROL STATEMENT

- The first statement of GPSS i/p is a control statement with the word SIMULATE in operational field.
- When a GPSS simulation run is finished the program does not immediately destroyed the model. Instead it looks for more i/p following the START statement keeping the model exactly it was at completion of the run.
- I/P following the START statement can charge the model. For example, A storage capacity would be define by increasing a 'storage' statement for assigning a new value.
- The model could also be modifying by changing existing blocks.
- Certain control statement can be included between START statement will wipe out all the statistics gathered so far, but will learn the system loaded with transaction.
- Another control statement, CLEAR, will not only wine wipe out the statistics of proceeding run but will also wipe out transaction in the system so that the simulation started from the beginning at the rerun.
- The END statement is used to terminate the run.

Example

SIMSCRIPT

SIMSCRIPT PROGRAM:

SIMSCRIPT is a very widely used language for simulating discrete system. A completely new version SIMSCRIPT II was released by RAND re-operation in 1968. The latest version of SIMSCRIPT II.5 is released in 1972. This language is very FORTRAN in appearance. SIMSCRIPT II.5 can be view as a general programming language with extra features for discrete event simulation. Because of this general power and FORTRAN based, SIMSCRIPT has been widely implemented and used discrete simulation language.

SIMSCRIPT SYSTEM CONCEPTS

The system to be simulated is considered to consist of entities having attributes that interact with activities. The interaction causes event that change the state of the system. In describing the system, SIMSCRIPT uses the terms, entities attributes, sets, event routine. An entity is a program element similar to a subscripted variable. When an entity is created it can be interpreted as a genetic definition for a class of entities. Individual entities have values all attributes which define particular state of the entity attributes are named not a number. For eg we may define employee to be an entity and AGE, SALARY are attributes.

Entities can be a 2 type

1) Permanent
2) Temporary

Permanent (specify for all time the proceeds)

Temporary (specify dynamically all the program proceeds)

Temporary entities are physically created and destroyed their special statements.

Permanent entities have their attributes stored as array

Both temporary and permanent entities can belong to sets. And own sets. SIMSCRIPT uses pointer to chain together entities that are members of sets. Commands are available to manipulate these sets. The user can define sets and facilities are provided for entering and removing entities into and prompt sets.

Activities are considered as extending over time with their beginning and their end being mark as events occurring instantaneously. Each type of event is described by a event routine, each of which is given a name and programmed as a separate closed sub routine. Activities may be endogenous and exogenous.

ORGANISATION OF A SIMSCRIPT PROGRAM:

Since event routines are closed routines some means must be provided for transferring control between them. It is done by the use of event notice.

These event notices are created when it is determine that event is scheduled.

An event notice exists for every endogenous events schedule to occur.

An event notice records

The time the event is due to

The event routine that is to execute the event

The general manner in which simulation proceed is illustrated in below fig.

TEMPORARY ENTITY RECORDS

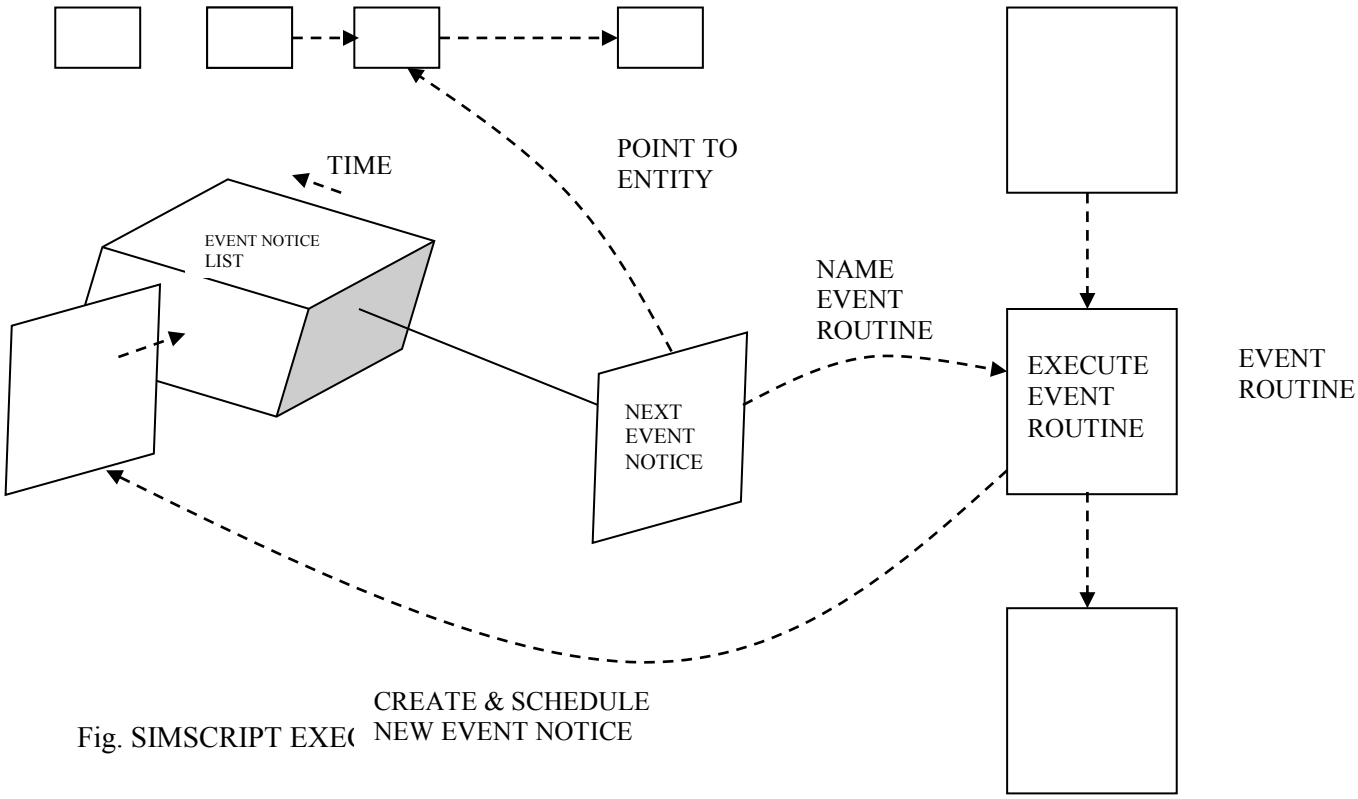


Fig. SIMSCRIPT EXE(

The event notices are filed in chronological order.

When all events (that can be executed at a particular time.) have been proceed the clock is update to the time next event notice and the control is passed to the event routine identify by the notice.

If the event executed by a routine results another events, the routine must create a new event notice and filed it with the other notice

In case of exogenous events a series of exogenous statements are created. For each event this statements are similar to event notices are also filed into chronological order and are read by the program.

SIMSCRIPT

SIMSCRIPT is a powerful and widely used programming language for **discrete-event simulation**. It was initially developed by the RAND Corporation in the 1960s and has since evolved into SIMSCRIPT II.5, which is considered a general-purpose programming language with additional features for simulation modeling.

History of SIMSCRIPT

- **Initial Development**:
  - SIMSCRIPT was first developed by the **RAND Corporation** in the early 1960s.
  - It was designed to simplify the process of writing simulation programs for discrete-event systems.
- **SIMSCRIPT II**:
  - A completely new version, **SIMSCRIPT II**, was released by RAND in **1968**.
  - This version introduced significant improvements, including a more flexible syntax and enhanced simulation capabilities.
- **SIMSCRIPT II.5**:
  - The latest version, **SIMSCRIPT II.5**, was released in **1972**.
  - It is widely regarded as a general-purpose programming language with specialized features for discrete-event simulation.

SIMSCRIPT System Concept

The system is modeled as a collection of **entities** that interact through **activities**. Entities can be **permanent** (exist throughout the simulation) or **temporary** (created and destroyed during the simulation). Events drive the simulation, and each event is handled by an **event routine**.

Example: Bank Simulation

- **Entities**:
  - Permanent: Bank tellers, customers in queue.
  - Temporary: Customer transactions.
- **Events**:
  - Customer arrival (exogenous event).
  - Transaction completion (endogenous event).

SIMSCRIPT Program Structure

A SIMSCRIPT program consists of three main parts:

a. Preamble

The preamble is a declarative section where all modeling elements (entities, attributes, sets, and events) are defined. It does not contain executable statements.

Example: Preamble for Bank Simulation

```simscript
PREAMBLE
  ENTITIES
    PERMANENT ENTITIES
      EVERY TELLER HAS A STATUS
    TEMPORARY ENTITIES
      EVERY CUSTOMER HAS A TRANSACTION_TIME
  EVENT NOTICES
    EVERY ARRIVAL HAS A CUSTOMER
    EVERY DEPARTURE HAS A TELLER
END
```

- **Explanation**:
  - TELLER is a permanent entity with an attribute STATUS (e.g., busy or idle).
  - CUSTOMER is a temporary entity with an attribute TRANSACTION_TIME.
  - ARRIVAL and DEPARTURE are event notices.

b. Main Program

The main program initializes the system and starts the simulation. It typically creates permanent entities and schedules the first event.

Example: Main Program for Bank Simulation

```simscript
MAIN
  CREATE TELLERS
  FOR EACH TELLER, SET STATUS = IDLE
  SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
  START SIMULATION
END
```

- **Explanation**:
  - Creates tellers and sets their initial status to IDLE.
  - Schedules the first customer arrival using an exponential distribution for interarrival times.
  - Starts the simulation.

c. Event Routines

Event routines define the behavior of the system when an event occurs. Each event type has a corresponding routine.

Example: Event Routines for Bank Simulation

```simscript
EVENT ROUTINE ARRIVAL
 CREATE A CUSTOMER
 SET TRANSACTION_TIME = NORMAL.F(MEAN_TRANSACTION_TIME, STD_DEV)
 IF A TELLER IS IDLE
  ASSIGN CUSTOMER TO TELLER
  SET TELLER STATUS = BUSY
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ELSE
  FILE CUSTOMER IN QUEUE
 ENDIF
 SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
END

EVENT ROUTINE DEPARTURE
 RELEASE TELLER
 SET TELLER STATUS = IDLE
 IF QUEUE IS NOT EMPTY
  REMOVE CUSTOMER FROM QUEUE
  ASSIGN CUSTOMER TO TELLER
  SET TELLER STATUS = BUSY
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ENDIF
END
```

- **Explanation**:
  - **ARRIVAL Routine**:
    - Creates a customer and assigns a transaction time.
    - If a teller is idle, the customer is assigned to the teller, and a DEPARTURE event is scheduled.
    - If no teller is available, the customer joins the queue.
    - The next arrival is scheduled.
  - **DEPARTURE Routine**:
    - Releases the teller and sets their status to IDLE.
    - If there are customers in the queue, the next customer is assigned to the teller, and a DEPARTURE event is scheduled.

Organization of a SIMSCRIPT Program

The organization of a SIMSCRIPT program revolves around the concept of **event-driven simulation**, where the system's behavior is modeled as a sequence of events that occur at specific points in time. Below is a detailed explanation of the key components and their roles in a SIMSCRIPT program, based on the provided content.

1. Event Routines

- **Definition**:
  - Event routines are **closed subroutines** that define the behavior of the system when a specific event occurs.
  - Each event type (e.g., customer arrival, transaction completion) has a corresponding event routine.
- **Purpose**:
  - Event routines handle the logic associated with an event, such as updating system state, scheduling new events, or modifying entity attributes.
- **Example**:

```simscript
EVENT ROUTINE ARRIVAL
 CREATE A CUSTOMER
 SET TRANSACTION_TIME = NORMAL.F(MEAN_TRANSACTION_TIME, STD_DEV)
 IF A TELLER IS IDLE
  ASSIGN CUSTOMER TO TELLER
  SET TELLER STATUS = BUSY
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ELSE
  FILE CUSTOMER IN QUEUE
 ENDIF
 SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
END
```

2. Event Notices

- **Definition**:
  - An **event notice** is a data structure that records information about a scheduled event.
  - It includes:
    - The **time** the event is due to occur.
    - The **event routine** to be executed.
    - The **temporary entity** involved (if applicable).

- **Purpose**:
  - Event notices are used to schedule and manage events in the simulation.
  - They ensure that events are executed in the correct chronological order.
- **Example**:
  - When a customer arrives, an event notice is created for the ARRIVAL event, specifying the time of arrival and the ARRIVAL event routine.

3. Event Scheduling and Execution Cycle

- **Event Scheduling**:
  - When an event is scheduled, an event notice is created and filed in **chronological order**.
  - The event notice is added to the **event list**, which is sorted by event time.
- **Event Execution**:
  - The simulation proceeds by processing events in the order of their scheduled times.
  - The simulation clock is updated to the time of the next event, and control is passed to the corresponding event routine.
- **Automatic Control Transfer**:
  - The transfer of control between event routines is handled automatically by the SIMSCRIPT runtime system.
  - Programmers do not need to explicitly manage the flow of control between events.

4. Endogenous vs. Exogenous Events

- **Endogenous Events**:
  - These are **internal events** that arise from actions within the system.
  - Example: A customer completing a transaction (departure event).
  - Endogenous events are scheduled by event routines during the simulation.
- **Exogenous Events**:
  - These are **external events** that arise from actions outside the system.
  - Example: A customer arriving at the system (arrival event).
  - Exogenous events are typically defined at the start of the simulation and are read from a predefined schedule.

5. Event List Management

- **Chronological Order**:
  - Event notices are stored in a list sorted by event time.
  - The simulation clock advances to the time of the next event, and the corresponding event routine is executed.
- **Creating New Events**:
  - When an event routine executes, it may create new events (e.g., scheduling a departure after an arrival).
  - New event notices are created and added to the event list.
- **Automatic Processing**:
  - The SIMSCRIPT runtime system automatically processes events in the correct order, ensuring that the simulation progresses logically.

6. Example: Event Notice Lifecycle

1. **Initialization**:
   - The simulation starts by scheduling the first exogenous event (e.g., customer arrival).
   - An event notice is created and added to the event list.
2. **Event Execution**:
   - The simulation clock advances to the time of the first event.
   - The corresponding event routine (e.g., ARRIVAL) is executed.
3. **New Event Creation**:
   - The event routine may create new events (e.g., schedule a departure).
   - New event notices are added to the event list.
4. **Simulation Progress**:
   - The simulation continues by processing events in chronological order until no more events remain.

7. Example: Bank Simulation

Below is an example of how event notices are used in a bank simulation:

Preamble:

```simscript
PREAMBLE
  ENTITIES
    PERMANENT ENTITIES
      EVERY TELLER HAS A STATUS
    TEMPORARY ENTITIES
      EVERY CUSTOMER HAS A TRANSACTION_TIME
  EVENT NOTICES
    EVERY ARRIVAL HAS A CUSTOMER
    EVERY DEPARTURE HAS A TELLER
END
```

Main Program:

```simscript
MAIN
  CREATE TELLERS
  FOR EACH TELLER, SET STATUS = IDLE
  SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
  START SIMULATION
END
```

Event Routines:

```simscript
EVENT ROUTINE ARRIVAL
  CREATE A CUSTOMER
  SET TRANSACTION_TIME = NORMAL.F(MEAN_TRANSACTION_TIME, STD_DEV)
  IF A TELLER IS IDLE
    ASSIGN CUSTOMER TO TELLER
    SET TELLER STATUS = BUSY
    SCHEDULE A DEPARTURE IN TRANSACTION_TIME
  ELSE
    FILE CUSTOMER IN QUEUE
  ENDIF
  SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
END

EVENT ROUTINE DEPARTURE
  RELEASE TELLER
  SET TELLER STATUS = IDLE
  IF QUEUE IS NOT EMPTY
    REMOVE CUSTOMER FROM QUEUE
    ASSIGN CUSTOMER TO TELLER
    SET TELLER STATUS = BUSY
    SCHEDULE A DEPARTURE IN TRANSACTION_TIME
  ENDIF
END
```

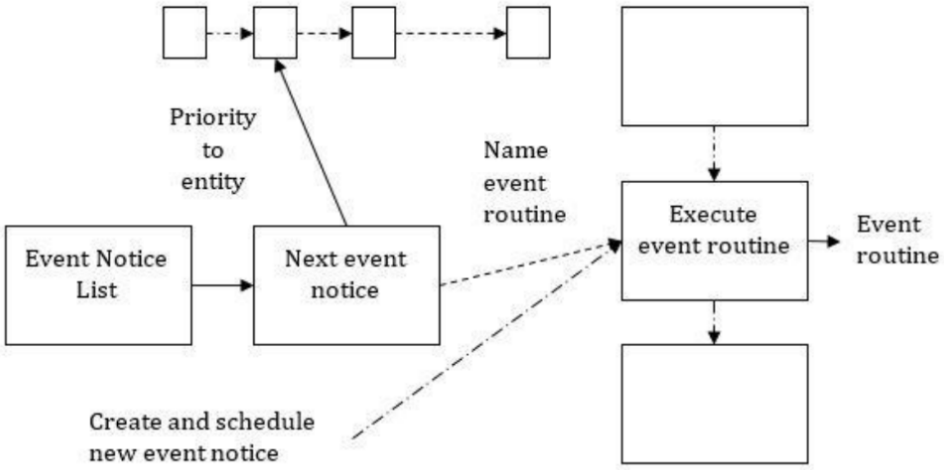


Fig: Organization of SIMSCRIPT Program

Key Features of SIMSCRIPT

SIMSCRIPT II.5 combines the power of a general-purpose programming language with features specifically designed for simulation. Below are its key features:

a. FORTRAN-like Syntax

- SIMSCRIPT's syntax is similar to **FORTRAN**, making it easy for programmers familiar with FORTRAN to learn and use.
- Example:

```
simscript

DEFINE X, Y AS REAL VARIABLES
SET X = 10.5
SET Y = X + 5.0
```

b. Discrete-Event Simulation Capabilities

- SIMSCRIPT is specifically designed for **discrete-event simulation**, where the system state changes at discrete points in time.
- It provides built-in constructs for defining **entities**, **attributes**, **events**, and **sets**.

c. Entities and Attributes

- **Entities**:
  - Represent objects in the system (e.g., customers, machines, or resources).
  - Can be **permanent** (exist throughout the simulation) or **temporary** (created and destroyed during the simulation).
- **Attributes**:
  - Properties of entities (e.g., `STATUS` of a machine, `TRANSACTION_TIME` of a customer).

d. Events

- **Events** represent occurrences that change the state of the system.
  - **Endogenous Events**: Arise from within the system (e.g., a customer completing a transaction).
  - **Exogenous Events**: Arise from outside the system (e.g., a customer arriving).
- Each event is handled by an **event routine**, which defines the system's behavior when the event occurs.

e. Sets

- **Sets** are collections of entities (e.g., a queue of customers waiting for service).
- SIMSCRIPT provides built-in operations for managing sets, such as adding, removing, and searching for entities.

f. Event Scheduling

- SIMSCRIPT uses **event notices** to schedule and execute events in chronological order.
- The simulation clock advances to the next event time, and the corresponding event routine is executed.

g. General-Purpose Programming

- SIMSCRIPT II.5 can be used as a general-purpose programming language, with support for:
    - Variables (real, integer, text).
    - Arrays and tables.
    - Control structures (e.g., loops, conditionals).
    - Input/output operations.

Applications of SIMSCRIPT

SIMSCRIPT is widely used in various fields for simulating discrete systems. Some common applications include:

1. **Telecommunications**:
    - Modeling call centers, network traffic, and communication protocols.
2. **Manufacturing**:
    - Simulating production lines, inventory systems, and supply chains.
3. **Transportation**:
    - Modeling traffic flow, airport operations, and public transit systems.
4. **Healthcare**:
    - Simulating patient flow, hospital operations, and resource allocation.
5. **Military and Defense**:
    - Modeling logistics, troop movements, and battlefield scenarios.

Advantages of SIMSCRIPT

- **Flexibility**:
    - SIMSCRIPT can model complex systems with ease due to its powerful simulation constructs.
- **FORTRAN-like Syntax**:
    - Familiar syntax makes it easy for programmers to learn and use.
- **General-Purpose Capabilities**:
    - Can be used for both simulation and general-purpose programming.

- **Wide Adoption**:
    - SIMSCRIPT has been widely implemented and used in academia, research, and industry.

Limitations of SIMSCRIPT

- **Outdated**:
    - SIMSCRIPT II.5 was released in 1972 and may not be compatible with modern systems.
- **Limited Community Support**:
    - The user community for SIMSCRIPT is small compared to modern simulation tools.
- **Learning Curve**:
    - While the syntax is FORTRAN-like, the simulation-specific constructs may require additional learning.

Example: Bank Simulation

A complete SIMSCRIPT program for a bank simulation:

```simscript
PREAMBLE
  ENTITIES
    PERMANENT ENTITIES
      EVERY TELLER HAS A STATUS
    TEMPORARY ENTITIES
      EVERY CUSTOMER HAS A TRANSACTION_TIME
  EVENT NOTICES
    EVERY ARRIVAL HAS A CUSTOMER
    EVERY DEPARTURE HAS A TELLER
END

MAIN
  CREATE TELLERS
  FOR EACH TELLER, SET STATUS = IDLE
  SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
  START SIMULATION
END

EVENT ROUTINE ARRIVAL
  CREATE A CUSTOMER
  SET TRANSACTION_TIME = NORMAL.F(MEAN_TRANSACTION_TIME, STD_DEV)
  IF A TELLER IS IDLE
    ASSIGN CUSTOMER TO TELLER
    SET TELLER STATUS = BUSY
```

```
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ELSE
  FILE CUSTOMER IN QUEUE
 ENDIF
 SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
END

EVENT ROUTINE DEPARTURE
 RELEASE TELLER
 SET TELLER STATUS = IDLE
 IF QUEUE IS NOT EMPTY
  REMOVE CUSTOMER FROM QUEUE
  ASSIGN CUSTOMER TO TELLER
  SET TELLER STATUS = BUSY
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ENDIF
END
```

SIMSCRIPT Concepts with Examples

Below is a detailed demonstration of the concepts mentioned, including **names and labels**, **SIMSCRIPT statements**, and **defining the system**, with examples to illustrate each concept.

1. Names and Labels

Names

- **Rules**:
  - Can consist of letters, digits, and periods (.).
  - Must contain at least one letter.
- **Examples**:

```simscript
AGE        ; Valid
AB12       ; Valid
PERSON.AGE   ; Valid
123AGE     ; Invalid (starts with a digit)
```

Labels

- **Rules**:
  - Used to identify programming statements.
  - Enclosed in single quotation marks.
- **Examples**:

```simscript
'LABEL1'    ; Valid
```

```
'AB12'      ; Valid
'1234'      ; Valid
```

2. SIMSCRIPT Statements

Print Statements

- **Syntax**:
  - PRINT n LINES AS FOLLOWS
  - PRINT n LINES THUS
  - PRINT n LINES WITH X AND Y LINE THIS (prints variable values).
- **Examples**:

```simscript
PRINT 1 LINE AS FOLLOWS
"Customer arrived at time", CLOCK

PRINT 1 LINE WITH X AND Y LINE THIS
"X =", X, "Y =", Y
```

Formatting

- **Three-digit integer**: ***
- **Four-digit real number with one decimal place**: ***.*
- **Example**:

```simscript
PRINT 1 LINE WITH X AND Y LINE THIS
"X =", X USING "***", "Y =", Y USING "***.*"
```

3. Defining the System

Preamble Section

- The preamble defines the system structure and simulation conditions.
- **Example**:

```simscript
PREAMBLE
 ENTITIES
  PERMANENT ENTITIES
   EVERY TELLER HAS A STATUS
  TEMPORARY ENTITIES
   EVERY CUSTOMER HAS A TRANSACTION_TIME
 EVENT NOTICES
  EVERY ARRIVAL HAS A CUSTOMER
  EVERY DEPARTURE HAS A TELLER
END
```

Variable Types

- **Types**:
  - REAL: Floating-point numbers.
  - INTEGER: Whole numbers.
  - ALPHA: Alphabetic characters.
  - TEXT: Character strings.
- **Default Mode**:
  - Normally, the mode is INTEGER or ALPHA.
- **Example**:

```
simscript

DEFINE X AS A REAL VARIABLE
DEFINE Y AS AN INTEGER VARIABLE
DEFINE NAME AS AN ALPHA VARIABLE
DEFINE ADDRESS AS A TEXT VARIABLE
```

Global Variables

- **Syntax**:
  - THE SYSTEM HAS X AND Y
- **Example**:

```
simscript

THE SYSTEM HAS MEAN_INTERARRIVAL_TIME AND MEAN_TRANSACTION_TIME
```

DEFINE Statement

- **Syntax**:
  - DEFINE name1, name2, ..., AS INTEGER VARIABLES
- **Example**:

```
simscript

Copy

DEFINE COUNTER, TOTAL AS INTEGER VARIABLES
```

Entities and Event Notices

- **Syntax**:
  - PERMANENT ENTITIES
  - TEMPORARY ENTITIES
  - EVENT NOTICES
- **Example**:

```
simscript

TEMPORARY ENTITIES
  EVERY CUSTOMER HAS A TRANSACTION_TIME
EVENT NOTICES
```

```
  EVERY ARRIVAL HAS A CUSTOMER
```

Arrays and Tables

- **Syntax**:
  - DEFINE table AS AN INTEGER, n-DIMENSIONAL VARIABLE
- **Example**:

```simscript
DEFINE WAIT_TIMES AS A 1-DIMENSIONAL REAL VARIABLE
```

Example: Bank Simulation

Below is a complete SIMSCRIPT program that demonstrates all the concepts mentioned above:

```simscript
PREAMBLE
 ENTITIES
  PERMANENT ENTITIES
   EVERY TELLER HAS A STATUS
  TEMPORARY ENTITIES
   EVERY CUSTOMER HAS A TRANSACTION_TIME
 EVENT NOTICES
  EVERY ARRIVAL HAS A CUSTOMER
  EVERY DEPARTURE HAS A TELLER
 VARIABLES
  DEFINE MEAN_INTERARRIVAL_TIME AS A REAL VARIABLE
  DEFINE MEAN_TRANSACTION_TIME AS A REAL VARIABLE
  DEFINE STD_DEV AS A REAL VARIABLE
END

MAIN
 CREATE TELLERS
 FOR EACH TELLER, SET STATUS = IDLE
 SET MEAN_INTERARRIVAL_TIME = 5.0
 SET MEAN_TRANSACTION_TIME = 4.0
 SET STD_DEV = 1.0
 SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
 START SIMULATION
END

EVENT ROUTINE ARRIVAL
 CREATE A CUSTOMER
 SET TRANSACTION_TIME = NORMAL.F(MEAN_TRANSACTION_TIME, STD_DEV)
 IF A TELLER IS IDLE
  ASSIGN CUSTOMER TO TELLER
  SET TELLER STATUS = BUSY
  SCHEDULE A DEPARTURE IN TRANSACTION_TIME
 ELSE
  FILE CUSTOMER IN QUEUE
 ENDIF
```

```
  SCHEDULE AN ARRIVAL IN EXPONENTIAL.F(MEAN_INTERARRIVAL_TIME)
  PRINT 1 LINE WITH CLOCK AND TRANSACTION_TIME LINE THIS
  "Customer arrived at time", CLOCK USING "***.*", "Transaction time", TRANSACTION_TIME USING "***.*"
END

EVENT ROUTINE DEPARTURE
  RELEASE TELLER
  SET TELLER STATUS = IDLE
  IF QUEUE IS NOT EMPTY
    REMOVE CUSTOMER FROM QUEUE
    ASSIGN CUSTOMER TO TELLER
    SET TELLER STATUS = BUSY
    SCHEDULE A DEPARTURE IN TRANSACTION_TIME
  ENDIF
  PRINT 1 LINE WITH CLOCK LINE THIS
  "Customer departed at time", CLOCK USING "***.*"
END
```

Explanation of the Example

1. **Preamble**:
    - Defines entities (tellers and customers), event notices, and variables.
2. **Main Program**:
    - Initializes tellers, sets simulation parameters, and schedules the first arrival event.
3. **Arrival Event Routine**:
    - Creates a customer, assigns a transaction time, and assigns the customer to a teller if available.
    - If no teller is available, the customer is filed in the queue.
    - Prints the arrival time and transaction time.
4. **Departure Event Routine**:
    - Releases the teller and assigns the next customer from the queue if available.
    - Prints the departure time.

SIMSCRIPT vs. GPSS

Feature	SIMSCRIPT	GPSS
Programming Style	Procedural, event-oriented.	Block-oriented, transaction-based.
Entities	Permanent and temporary entities.	Transactions.
Resources	Explicitly defined as entities.	Facilities and storages.
Events	User-defined event routines.	Implicit in block flow.
Control	User controls event scheduling and logic.	GPSS handles timing and logic automatically.
Statistics	User must collect and report statistics.	Automatic statistics collection.
Flexibility	Highly flexible, suitable for complex models.	Less flexible, easier for simple models.
Learning Curve	Steeper due to procedural nature.	Easier due to abstract nature.