Mine Design Using Simulation

JOHN R. STURGUL

Professor of Mining Engineering at the University of Idaho

CONTENTS

John R. Sturgul's snail mail address is:

Dr. John R. Sturgul
Dept. of Met. & Mining
Univ. of Idaho
Moscow, ID 83844-3024
ph: (208) 885 7939
FAX: (208) 885 2855

You may wish to send John R. Sturgul e-mail. This is much better than snail mail.

sturgul@uidaho.edu

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul

Chapter 1
INTRODUCTION TO SIMULATION
A REVIEW OF SIMULATION MODELS

Simulation Models

The GPSS (General Purpose Simulation System) computer programming language is a special language that is used primarily to simulate discrete systems. A discrete system is one where at any given instant in time, a countable number of things can take place. Nearly all of the problems one encounters in the study of queuing theory can be represented by discrete systems. Some of these examples are:

- People entering a barber shop with a single barber. If the barber is busy, people wait in the waiting chairs until it is their turn.

- People entering a bank with multiple tellers. The customers may either form individual queues at each teller or wait in a single queue (known as a "quickline").

- Trucks working at a construction site where a single shovel loads each truck. The trucks travel to a dump area where they dump and then return to the shovel. This is an example of a "cyclic" queue. The elements of the system, in this case, the trucks do not leave the system.

- Ships entering a harbor with multiple berths. The ships need to be towed into a berth with a tug boat or tug boats.

- Telephone calls arriving at a central switchboard where they need to be routed to the correct extension.

- Television sets on a conveyor belt arriving at a inspection station. If the set fails inspection, it may be sent back for adjustment or, in the worst case, it is discarded.

A complete treatment of simulation theory is beyond the scope of this book. However, an understanding of how simulation models are constructed and what they tell us is not too difficult.

Consider a bank with customers arriving and tellers giving service. All the possible events that take place in the bank are discrete events or can be considered as being such. Possible events might be customers arriving, customers joining a queue if all the tellers are too busy, customers going to

a teller who is free, customers leaving the bank when finished. Perhaps some of the customers will leave the queue if the waiting time is too long and go to another store or stores and return later. In most cases we shall be modeling systems that will involve some queueing such as will happen when all the tellers are busy in the bank, all the petrol pumps in a petrol station are being used, all the checkout counters at the grocery store are in use, etc.

GPSS/H is excellent for simulating systems that have this type of queueing. As we shall see, it is very easy to model a great variety of very complicated systems using GPSS/H.

What Will Be Modeled

The models we shall be studying might represent the bank working over a period of many months, an assembly plant that manufactures television sets, a barber shop where customers can obtains haircuts, shampoos, and manicures, or even a person doing her Saturday morning shopping. In some cases, the model may be only a small part of a large system such as the tool crib in a large factory.

The models we construct will not solve any problem directly but provide information about how the system is working and then how it will work with certain selected parameters changed. Suppose a company has their own fleet of cars for their salesmen to use. If the cars need any service, whether it is of a routine nature or major repairs, it is done by one of two mechanics. The company is concerned that the mechanics are not able to keep up with the repairs and wonders if it would be worth their while to hire another mechanic. Before the simulation model can be constructed the company must define the problem to be solved in greater detail than has been given here. The following information is also needed:

1. The company needs complete records of all services for each type of car. This includes the frequency of service and the distribution of times for the particular service.

2. They need to know what it would cost to hire a third mechanic as well as the cost in lost sales when a car is not available.

3. The fact that cars require routine maintenance or very minor repairs are given preference over those that are in for major service. This means that these cars are put in the front of the queue of any other cars that are waiting for service.

4. When the sales manager brings his car in for service, it is given a special status and this car is immediately worked on. Thus, even if both mechanics are busy, one will put aside the car he is working on and start the repairs on the manager's car.

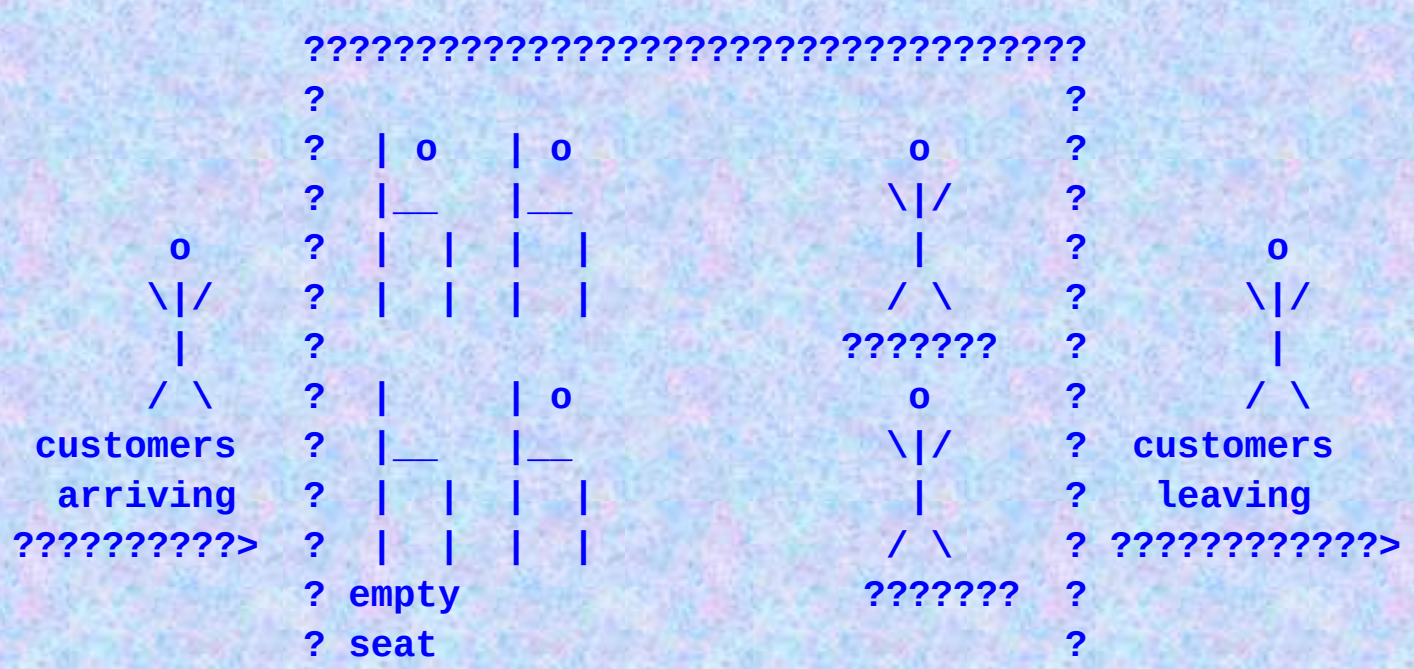
This information obtained from the simulation model might include the following:

1. How the system presently works. Obviously, the computer model has to accurately reflect the system as it is working before any reliability can be associated with the results from the changed model.

2. How the system works with selected changes, such as the car repair facility with three mechanics.

Another Type of Problem to be Studied and the Method of Solution

Consider the following system consisting of a two person barber shop with customers arriving to have their hair cut. The shop operates 8 hours a day. There are two chairs for the haircuts and only 4 chairs for the customers to wait if both barbers are busy. Thus, the system can only hold 6 customers. If a seventh customer arrives and finds the shop full, he will always leave. Both barbers are identical so it can be assumed that they work at the same rate and the customers have no preference for either barber. They are served on a first come, first served basis. However, customers do not like to be kept waiting too long and, so, if a customer finds he has been waiting too long, he will leave the barber shop. The barbers know this and so the time to give haircuts is a function of how many people are waiting, i. e., as more people are waiting, they will give haircuts faster. The customers do not arrive at regular intervals and the haircuts are not given at the same time. Both arrival rate and haircut rates are given by known statistical distributions, with the barbers having separate haircut rates. These depend on the number of customers waiting. The more customers waiting, the quicker the barber work. Figure 1.1 illustrates the situation of the barber shop.



??????????????????????????????????

**Figure 1.1 Two barbers, four chairs.
Both barbers are busy and three of the chairs are occupied.**

The owner of the barber shop would like to study the shop to see if it would be profitable to add another barber or simply add another chair for customers to wait. Perhaps it would be possible to purchase new equipment so that the barbers can work even faster. Would the extra haircuts justify the expense of this equipment? GPSS will assist in building a model to:

1. Predict how the system as outlined above works. The model will be only as good as the input data and the assumptions given above.

2. Once the model represents the barber shop as it presently is working, it can be modified to predict how it will work under different conditions.

Item 2. is where GPSS/H is so handy. As we will learn, changes in GPSS/H programs are often made by changing only a few lines of code. The fact that GPSS/H programs can be so easily changed to reflect the "What if?" type questions a person may want to pose, makes it an ideal language to use for simulation studies.

Once the modeler is satisfied that the original model is correct, the simulation can be re-done, but this time with the system having 3 barbers. Alternately, the model can be run for five seats for customers. Finally, the model can be run for different combinations of speeds for the barbers to work at.

By using the cost data for the various combination of barbers, lost customers, profit per haircut, etc, the modeler can then determine the economics of the system and make the correct choice.

A Simple Simulation Model

The following example will illustrate a situation of a simulation model with constant arrival rates and constant service rates. Suppose a tool crib has 1 attendant to serve a large group of machinists. These machinists come for a tool (one only) at a uniform rate of 1 every 5 minutes. It takes exactly 6 minutes to obtain the tool. Machinists earn $8/hour and the tool crib attendant earns $6/hour. The factory works an 8 hour shift but stops for a 1 hour lunch break. The crib is closed for lunch and the end of the 8 hour shift. In order to simplify the calculations, if a mechanic is waiting for a tool either at the lunch break or the end of the day, he or she will wait and be served. The tool crib operator does not receive extra pay for working overtime but the mechanics can deduct the time waiting from their actual working time. Should the company hire another tool crib attendant?

Solution

This is a very simple situation and one that would rarely be encountered in practice. Even so, it will prove instructive to learn what simulation models can tell us. The problem will be solved first for 1 tool crib attendant, then 2 and then 3.

The machinists arrive every 5 minutes, so there will 12 per hour arriving. In a 4 hour time period, 48 will arrive. The first arrives 5 minutes after the tool crib opens. There will be no wait for him. The second person arrives 5 minutes later and will experience a 1 minute wait, until the attendant is free. Similarly, the third person has a 2 minute wait, etc., up to the 48th person who has a 47 minute wait. In a 4 hour period there will thus be a total waiting time for the machinists of 1 + 2 + 3 + ... + 47 or 1128 minutes at $8/hour. This represents a loss of 1128/60 x $8 or $150.40 . For the two 4 hour periods in a day this represents a loss of $300.80. If two (or more) tool crib attendants are working, there will never be a wait for a free attendant (only the 6 minute wait for the tool). Table 1.1 summarizes the results from considering the cases of three attendants.

Table 1.1 Tool Crib Simulation

Number of Attendants	1	2	3
Number of machinists who arrive	48	48	48
Total time waiting for free attendant	1128	0	0
Cost of lost time/4 hours	$150.40	0	0
Pay to tool crib attendant/4 hours	$24.00	$48.00	$72.00
Total cost/8 hours	$324.80	$96.00	$144.00

Clearly, it would be advantageous to hire one additional attendant. The result of this simple model may or not be useful to a company, depending on the original hypotheses, which were:

1. Arrival rate of one mechanic every 5 minutes. In practice, the arrival rate will be at random times. There may be an average rate of so many per hour but, in general, the time of arrival for a particular mechanic will be random.

2. Service rate of the tool crib attendants is constant. Here, too, in practice, the service rate

will normally be random.

3. The tool crib closed every 4 hours. Anyone waiting for service immediately left when the 4 hours was up. In practice, the mechanics about to be served will still obtain service.

4. The length of the queue tended to grow to an eventual size of 8 every 4 hours. The is not realistic. If the queue is too long an arriving mechanic will tend to leave and come back later when the line is shorter.

5. Each arriving mechanic wanted only 1 tool. In practice, the number of tools needed may be 2, 3, or more.

It will be shown that, using the GPSS language, a model can easily be constructed to include all of the above possible changes to the original assumptions. Problems, such as the one discussed so far are quickly and easily solved with GPSS.

Review of Queueing Theory

The example of the queueing problem for the tool crib has an exact solution. There are very few such solutions available, especially for problems involving cyclic queues for a finite population. Cyclic queues are those where the system under study has elements that do not leave, such as trucks working in a quarry. Here the trucks are loaded, haul, dump and return to the loader. Whenever there is a finite population, as soon as one element is doing a particular thing, the statistical distribution governing rates will change. Thus, if a company has a fleet of 10 cars to be studied, if two are being serviced the probability of another one coming for service is no longer the same as when all 10 were up and running.

In general, in order to study complex systems where queueing takes place, it is necessary to build computer simulation models. It will be shown that GPSS is an ideal computer programming language to model such systems. In fact, one of the features of GPSS is that, as one learns the language, one automatically learns how to build complex simulation models. In the next Chapter we shall write our first GPSS program, but first it might be instructive to review a few basic concepts from queueing theory. These have to do with the possible arrival distribution, service distributions, number of servers either in series or parallel, the population size and the queue discipline. Table 1.2 gives the possibilities to consider.

Table 1.2 Possibilities for Queueing System

1. Population:
Infinite
Finite

2. Arrival time distribution:
- **Constant**
- **Poisson**
- **Erlang**
- **Uniform**
- **Arbitrary**
- **Normal**

3. Service time distributions:
- **Exponential**
- **Erlang**
- **Constant**
- **Uniform**
- **Arbitrary**
- **Changing with the time of the day or queue length**

4. Service facilities
- **Single**
- **More than one in parallel**
- **More than one in series**
- **Variable number, both in parallel and in series**

5. Queue discipline
- **FIFO (first in - first out)**
- **Random**
- **LIFO (last in - first out)**
- **Priority of one type of customer over another for the position in the queue**
- **Ability of one customer to preempt another one being served**
- **Priority of shortest or longest service time being served first or last**
- **Balking (customer refuses to join if queue too long)**
- **Switching from queue to queue**
- **Leaving (customers will leave if waiting too long)**
- **Being a member of more than one queue (a person can be in a shopping center and take a number for meat service at the same time as she is waiting for her vegetable number to be called).**

It may come as a surprise to the person who has not formally studied queueing theory but it is not possible to obtain exact solutions to all of the above situations (although a lot of very fine mathematicians have tried). However, several problems do have solutions and these can be found in textbooks on Operations Research or Queueing Theory. As one learns how to construct simulation models, it is instructive to compare the results from the simulation model with what one expects to obtain from an exact solution.

Simulation vs. Mathematical Solution

To illustrate a comparison of a simulation model with one that has an exact solution, consider the case of a store where customers arrive on the average of 24 per hour. The arrivals are Poisson. The single clerk in the store can handle a customer on the average of 1 every 2 minutes. The distribution for this service is exponential. Service is first come - first served. The customers do not mind waiting if there is a queue. It is desired to simulate the store for 50 days or 10 weeks of operation, where a day is 8 hours in duration and the store operates continuously. Compare the results with those obtained by an exact solution.

Solution

The problem will be recognized as a standard one that is discussed in any text on queueing theory (See, for example, Operations Research by Phillips, D. T., Ravindran, A. and Solberg, J. J., John Wiley and Sons, New York, 1976, chapter 7). The exact mathematical solution is available and equations can be found for determining the probability of the clerk being idle, the probability of any number of customers being in the store, the expected number of customers in the store, the average time for a customer to wait in the queue, to be in the store, etc.

Even though an exact solution exists to this problem, a computer program was written in GPSS/H to illustrate the way one will use the language to solve such problems. The simulation model used Monte Carlo simulation. This technique uses a random number generator to simulate both arrival times and service times. The simulation starts at simulated time t = 0 and will run until the program reaches a point in simulated time that the programmer feels is enough to yield correct results. First, a basic time unit needs to be selected. This is normally taken as the smallest time as given by the statement of the problem. For the example here, a time unit of 1 minute is selected. Thus, the customers will arrive on the average of every 2.5 time units. The clerk can handle a customer every 2 time units. The simulation is then done for times of 8, 50, 100, 200, 400, etc. hours. These have to be converted to minutes since the basic time unit is a minute.

Since the exact solution assumes steady state conditions, the simulation is run for 4 hours (240 time units) and is then stopped. All relevant statistics, except for the customers in the system, are discarded. Then, the simulation is restarted and run for the desired simulated time. Selected portion of the output from this program for the simulated time of 400 hours are:

Customers serviced	9605
Percent time clerk busy	.801
Average number of customers in system	4.122
Average time in system, minutes	10.2

The theoretical values can be found by use of simple formulas which can be found in any book

on Operations Research or Queueing Theory.

Customers served	**9600**	**customers arrive on average of 24/hr. for 400 hours**
Percent time clerks busy	**.800**	**this is 24/30**
Avg. number of customers in system	**4**	
Average time in system	**10 minutes**	

As can be seen, the above results compare quite favorably with those obtained by the simulation. It is important in both interpreting and using the results of a simulation that the simulation has been allowed to run for a long enough period or the results may not be accurate. In performing a simulation, one would like to obtain results that can be reproduced nearly identically if other simulations are done with different random numbers. There is no set answer to the question of how many simulations are enough, as the proper number of time units to simulate for is a function of several variables. One is the nature of the simulation, i.e., is the population infinite or finite? In the case just considered of an infinite population and Poisson arrivals with exponential service, a large number of simulations have to be performed. In the case of a system where the parameters being simulated cycle through the system (such as workers in a factory) not quite so many simulations may be needed. The nature of the queue and the service facilities are also important. In addition, if the statistical distributions are relatively uniform, such as a normal distribution with small standard deviation, the simulations tend to achieve a level of stability rapidly. This last result is important (and comforting) for the person doing simulations who has a lot of data that is normally distributed. This is often the case for working times in a factory, truck haulage rates along a road, manufacturing times, etc. If the statistical distributions are non-symmetric to a large extent, the number of simulations to be performed can be great. This will be demonstrated by means of an example later in this Module. First, let us again consider the example just solved.

Suppose, however, that, for the simple queueing system just studied, the simulation was done for less than 400 hours. What would the results have been? The answer depends, in part, on the sequence of random numbers. But it is instructive to re-do the simulation for less than 400 hours and examine the results. Table 1.3 summarizes the results from these different simulations.

Table 1.3 Resutls of Re-running Simualtion for Store

Simulated time (hrs.)	customers served per hour	percent time clerk busy	avg. no. customers in system	avg. time customers in system

8	26.9	92.2	4.34	9.58
50	23.6	81.7	4.105	7.87
100	24.0	82.6	4.061	10.63
200	23.9	79.7	4.810	11.52
400	24.2	80.3	4.122	10.2
theorectical	24.0	80.0	4.00	10.2

As can be seen, the results for simulating for 8 hours are quite different from the theoretical ones. Simulating for 200 hours yields results that are becoming close to the theoretical ones, except for the average number of customers in the system. After 400 hours the simulated results are quite close to the expected ones. If this problem was for a real store, the simulation may well have run for an even longer time.

Simulation with Non-symmetric Distributions

Whenever the statistical distributions are non-symmetric (the Poisson is non-symmetric), the number of simulations may have to be very large. This is easy to understand, since it is desired to model a system over every possible situation and in theory, repeat the simulation until the various parameters being studied do not change. To illustrate this concept of non- symmetric distribution, consider a simple example. Suppose a person is modeling his behavior on a day to day basis, weekends not included. Each day this person stops at the local casino and bets $2 on number 7 on a roulette wheel. He makes only this bet and, whether he wins or loses, will leave. The probability of winning is 1/38 (the wheel has numbers running from 1 to 36 as well as a zero, 0, and double zero, 00). How many simulated days are needed to produce satisfactory results for the simulation? Certainly not 38, as the expected number of wins is only 1. How about 380 or 3800? To study this, a short GPSS program was written. The simulation was performed for 380, 3800, 38000 and, finally, 380,000 days. It was then run for three different sequences of random numbers. Table 1.4 summarizes the results of these three simulations.

Table 1.4 Results of Studying Simulation of Roulette Wheel

Number of days	First set of random nos. Observed wins	%error	Second set of random nos. Observed wins	%error	Third set of random nos. Observed wins	%error	Expected wins
380	8	-20.0%	13	+30.0%	5	-50.0%	10
3800	95	-5.0%	95	-5.0%	95	-5.0%	100
38000	1019	+1.9%	1019	+1.9%	1017	+1.7%	1000
380000	10024	+.2%	10029	+.2%	10021	+.21%	10000

As can be seen, the observed number of wins versus the expected number of wins start to approach each other only after a large number of simulations (it may be interesting to note here that the outcome of the simulation for all three cases of 380000 days shows a net loss of $57,336, as the casino pays out at a rate of 36 times the bet whenever the number 7 came up. It is only for the smaller number of simulations of 380 days that a gain can be found (second set of random numbers). While the outcome will be slightly different for each new simulation, one is soon convinced that, in the long run, the casino will always come out ahead.). In fact, the results for 380 simulations give results that vary from the expected number of wins by as much as 50%. Thus, for situations such as the above, one must always be aware that a very large number of simulations may be needed (there's no guarantee that even 380000 is enough depending on the problem!). Fortunately, for most situations the simulation can be successfully performed with a reasonable number of simulations.

Why Do a Simulation?

It always comes as a surprise to students to learn that it is rarely possible to obtain exact solutions to any but the most elementary problems that lead to a queueing situations. Even though we all understand queueing situations as we experience them daily whenever we enter a bank that has many customers and we have to wait for a teller, shop in a large grocery store and wait in the checkout line, etc. One would think that such problems can be solved quite easily but this is not the

case. Although the field of queueing theory has been studied by the Mathematicians for many years, very few problems have been solved.

A computer simulation can rapidly and accurately solve most elementary and complex situations where one encounters queueing situations.

What is Meant by a "Solution to a Queueing Problem?

When we obtain a solution to a queueing problem it is important to understand just what is meant by our solution. A simulation model does not solve a problem but tells us how a system will operate under a given set of parameters. For example, the model might tell you that if you have 9 trucks in your mine, the daily profit will be $457. Adding a 10th truck and doing another simulation might then tell us that the new profit will be $505. Doing a further simulation for 11 trucks might tell us that the new profit will be $480. Thus, we conclude that the optimum number of trucks to have in the mine is 10.

Another Example of a Simulation Model

Consider a simple example of a single shovel loading trucks at a construction site. This shovel can only load a single truck at a time. After each truck is loaded, it travels to a dump area where it dumps its load and then returns to the shovel. If the shovel is free (no other truck is being loaded), it immediately begins to be loaded again. If not, it waits in a queue until the shovel is free.

Assume that you are going to study this system (the trucks and shovel and the travel paths make up the system). You are told by the engineer in charge who has studied the shovel and the haulage routes that the shovel can load a truck in exactly 5 minutes (this is a slow shovel, but don't worry about this for the present). It takes exactly 8 minutes to drive to the dump, exactly two minutes to dump and exactly 6 minutes for a truck to return to the shovel.

If you had a single truck in the system, it would load in 5 minutes and then take 16 minutes to return to the shovel. The shovel would be busy for every 5 minutes out of 21 or 23.8% of the time. There would be a load of ore dumped every 21 minutes or approximately 3 per hour. In an 8 hour shift you would expect that there would be slightly less than 24 loads dumps (this construction site does not allow the workers any breaks).

If you added another truck to the system, you would expect the production to double and the shovel to be twice as busy. Adding a third truck would likewise increase production and the shovel would be busy about 72% of the time. What will happen when you add a 4th truck? The answer is that the system will experience no problems yet as at any one time there can be a truck being loaded (this takes 5 minutes) and the other 3 can be traveling to or from the dump. It is only when you have 5 trucks working that you start to have queueing problems. However, production will

increase by having 5 trucks as compared to having 4. The question is how much and will this be worth it? To answer this last question additional data is needed. Let us assume that each load carried by the trucks somehow represents a contribution to profit of $45 and that each truck costs $225/day to run.

Table 1.5 gives the results of a computer simulation for the above problem. The simulation was run for 10 "days" with each day being an 8 hour shift. At the start of the simulation all of the trucks were at the shovel and the trucks worked for 10 "days" straight. The results of the simulation are:

Table 1.5. Results of Computer Simulation

N	Loads	Shov. utl.	Avg Que	Profit per day
1	229	.239	0.000	805
2	458	.477	0.000	1611
3	686	.715	0.003	2412
4	914	.952	0.006	3213
5	959	1.000	0.807	3190
6	959	1.000	1.807	2965

It is easy to see that the number of trucks to have for optimum profit is 4. Note that adding a 5th truck will increase production but will not result in a greater profit.

A Change to the Problem

The problem just completed assumed that all of the times used in the model were constant. This is certainly not the case in real life. Things do not happen in exact times. The time to load a truck will vary depending on several parameters, the time to drive to the dump will not be constant, etc. Let us assume that you did some time studies at the construction side and found that the time to load a truck took an average of 5 minutes but the statistical distribution that best describes it is the exponential distribution. The travel times and the dumping times are best described as coming from normal distributions as follows:

travel to dump	mean 8 min	std dev 1.5 min
dump	mean 2 min	std dev .3

return to shovel	mean 6 min	std dev 1.15 min

As can be seen the mean times have remained the same.

The computer program was modified to allow for these changes and run for 100 days (the reason for this is because the exponential distribution was used - whenever this is used the amount of simulations or the simulated time is substantially increased - more on this later). Table 1.6 gives the results of the simulations:

Table 1.6 Results of Simulation

N	Loads	Util of shovel	avg queue	Profit
1	2281	.241	0.00	801
2	4355	.445	0.10	1509
3	6056	.627	0.35	2050
4	7515	.768	0.73	2481
5	8285	.876	1.36	2603
6	8948	.936	2.08	2676
7	9243	.974	2.94	2584
8	9434	.989	3.87	2405
9	9423	.997	4.84	2215
10	9547	.999	5.82	2046
11	9550	1.000	6.82	1820
12	9523	1.000	7.82	1585

Notice that the number of loads keep increasing as the number of trucks is increased until 10 trucks are working. Also note that the profit is a maximum for 6 trucks. This profit is $2676, which is considerably less than the previous profit of $3219. Thus, the number of trucks needed is 50% more than before and the optimum profit is 21% less. Also note that it really would not make much difference if 5 trucks were used rather than 6. Do you see what happens when too many trucks are in the system? Notice that the average queue length for 9 trucks is 4.84 and for each additional truck the average queue will increase by 1. This means that each additional truck will, in effect, add 1 to the average queue. No increase in production will result.

note: Although this seems like a simple model, this was studied by numerous investigators in the 1960's and 1970's to determine the optimum number of trucks to have for construction projects.

Numerous lengthy reports and papers were written on this problem alone.

Some comments on the GPSS language follow.

Why use the GPSS Language?

All of the examples in this book are solved using the GPSS/H simulation language. Since this is generally not the first computer language mining engineers have been trained to program in, it is appropriate to learn why GPSS was selected. In fact, there are multiple versions of GPSS available and the one used here is GPSS/H, which is designed specifically for the personal computer. This chapter is intended to answer questions about the GPSS language and will be done in a question and answer format.

What Is GPSS

GPSS (General Purpose Simulation System) is both a computer language and a computer program. It was designed for studying systems represented by a series of discrete events. A discrete system is one where only a countable number of events can occur at one time. These discrete events might be trucks being loaded, ships entering a harbor, people entering a bank, cars travelling on a road, parts on a conveyor belt, etc. GPSS is a high level, non-procedural language.

Where Did It Come From?

GPSS originally was developed by Geoffrey Gordon for IBM in the early 1960's and released to the public around 1963 or 1964 so it has been around for quite some time. However, it is a dynamic language in that new versions keep being introduced every three or four years. It is now a multi-vendor language and various versions are available. It is widely used on both main frames and PC's. By 1972 there were at least 10 versions of GPSS available and many of these have survived in one form or other. Greenberg (1972) presents details about these early versions of GPSS and traces their histories.

What About Modern Versions Of GPSS?

Since GPSS has been around for quite some time, it is natural that people who were introduced to it at an early stage of its development may not be aware of how it has changed. An excellent summary of recent developments is given by Schriber (1988). Schriber lists the common, modern versions of GPSS (GPSS/H, GPSS V, GPSS/PC, GPSSR/PC and GPSS/VX) and where to obtain relevant information regarding each. It is possible to add animation to the results of a simulation and so view the simulation in "cartoon" fashion. In fact, it is possible to find certain functions

performed by GPSS embedded in other languages such as GPSS-Fortran, APL-Fortran and PL/1-GPSS. Schriber's paper gives references for these.

Henriksen (1983 and subsequent company updates) dispels some of the myths that have grown up surrounding GPSS. These are:

1. **"GPSS is inherently slow". This is no longer the case. In fact, comparisons of GPSS with other simulation languages by Abed, Barta and McRoberts (1985a, 1985b) show that GPSS is many times faster than other languages such as SLAM and SIMSCRIPT.**
2. **"To do anything sophisticated in GPSS, reference to other languages is needed." This is rarely the case anymore. In fact, none of the examples presented in this book revert to any other computer language.**
3. **"GPSS is trivial to learn." Any computer language takes time and practice to master and GPSS is no exception. Most industrial short courses last 4 - 5 days by which time the participants have a sound introduction to the language. It is normally taught on the university level as a full semester course.**
4. **"Modeling difficulties arise more frequently due to language shortcomings than due to lack of modeler expertise". This is certainly not the case with modern versions of GPSS. Gordon (1978) and Henriksen (1983) give examples of people who blame GPSS for their lack of expertise in solving simulation problems when the real fault lies with their own lack of programming expertise.**

What Is A Non-procedural Language?

A non-procedural language is one that anticipates what the programmer is attempting to do and allows the computer code to be very short. Often the programming code for a non-procedural language appears very similar to the problem it has been designed to solve. For example, sorting an array of data using a procedural language is done in one of several ways. One way is to find the smallest (or largest) element, place this at the front of the number, and then sort through the remaining numbers for the next smallest (or largest), find it and place this second in the last, etc. until the array is sorted. This involves numerous comparisons of data. A non-procedural language that is used for handling data bases where it is common to sort data may have a single command, namely, SORT to do this. In simulation studies, one often encounters queues. To model a queue in GPSS to gather certain statistics, the single line of code (known as a Block) might look as simple as the following

QUEUE DUMP

No code for output is needed: GPSS will automatically gather relevant statistics and output them when the program is finished.

People seeing a GPSS program for the first time tend to remark, "Is that all that there is to it?" As we shall see, this is one of the remarkable features of the language. Since it was designed specifically for solving certain problems, it is indeed quite compact.

In the study of queueing theory one soon encounters a system having a single server for people arriving at random times from an infinite population. One case of this is known as the M/M/1 (the first M stands for Markov to indicate that the arrival rates are Poisson, the second M for exponential server and the 1 to indicate a single server). This is modeled in GPSS by writing only 7 (!) lines code, which are known as programming Blocks. The equivalent Fortran program would take many hundreds of lines of code. In addition, to make changes in a GPSS program to answer the "What if?" questions often takes only a few lines of code. If a system is being studied with room for only 8 trucks working, the relevant line of code may be:

```
STORAGE S(TRUCKS),8
```

To study the same system but with room for 9 trucks may involve changing only the above line to:

```
STORAGE S(TRUCKS),9
```

Is GPSS Hard To Learn?

GPSS is not any more difficult to learn than any other programming language. Most people find it easier to learn than traditional engineering languages such as Fortran, Basic or Pascal. After about 30 or 40 hours of instruction, most engineers find that they can proceed on their own with writing practical simulation programs. Since it is a very popular language, numerous short courses are held throughout the world.

Will A Knowledge Of Other Languages Such As Fortran Help To Learn GPSS?

Not really. The logic behind GPSS is so different, knowledge of other procedural languages may even be a hindrance. Of course, knowledge of any other simulation language is a different matter.

What About Using Other Simulation Languages?

Other simulation languages exist that are quite good for solving simulation problems relating to mining. Some of these are: SIMAN, SIMCRIPT II.5 and SLAM. The solutions obtained by investigators using these languages may well be as accurate as those obtained by GPSS. However,

GPSS/H was selected as the language for this manual for the following reasons:

1. **It is multi-vendor so it is continually being upgraded.**
2. **It is widely available.**
3. **It is written in machine language and, therefore, is inherently very fast.**
4. **It can solve a wide variety of problems rapidly and accurately. These problems come from many sectors such as manufacturing, engineering, business, science, etc.**
5. **It has withstood the test of time, having been introduced by IBM in 1961. Other simulation languages have fallen by the wayside.**

Will GPSS Replace Languages Such As Fortran, Pascal Or Basic?

No. There are a large number of problems that should and will always be solved using traditional computer languages. (This does not exclude the possibility of having packages available to solve mining problems based on traditional languages). Learning GPSS enhances a person's computer skills rather than replacing any. In this regard it can be looked upon as adding a computer skill such as word processing or learning how to construct a spread sheet. Knowledge of these does not replace programming skills using traditional procedural languages.

How About A Comparison Between Fortran And GPSS For A Simulation Study?

Below is a rough comparison. The actual values will depend on the particular problem. This might be for the simulation study of the ships entering an average sized harbor.

Comparison of GPSS with Fortran

	GPSS	Fortran
time to write program:	1 - 2 days	many months
execution time:	< 1 min CPU	3 - 4 hours CPU (386)
ease of changing program:	trivial	up to a week
lines of computer code:	300 - 400	20000 - 50000
user friendly?:	yes	rarely
graphical output	few lines of code	many additional lines of code

animation?:	available	not standard

But Aren't There Fortran Packages On The Market?

Yes, but these tend to be very expensive (some around $50,000), hard to change, not user friendly and take a great deal of CPU time to run.

But Why GPSS/H?

Of all the version of GPSS available this is by far the most advanced. It contains features that the other versions of GPSS do not have. However, learning GPSS/H does not exclude person from using other versions. Conversion from one version to another is not at all difficult. However, many of the features of GPSS/H will not work on other version.

References

Abed, S. Y., Barta, T. A. and McRoberts, K. L. (1985) "A Qualitative Comparison of Three Simulation Languages: GPSS/H, SLAM, SIMSCRIPT." Computers & Indus. Engr. no 9, 35-43.

Abed, S. Y., Barta,T. A. and McRoberts, K. L., (1985) "A Quantitative Comparison of Three Simulation Languages: GPSS/H, SLAM, SIMSCRIPT." Computers & Inds. Engr. no. 9 45 -66.

Greenberg. S. (1972). GPSS Primer Wiley-Interscience, New York.

Henriksen, J. O., (1983) "State-of-the-art GPSS" In:Proc. of the 1983 Summer Computer Simulation Conference The Soc. for Com. Sim., San Diego, CA 918 -913 (Wolverine Software has published updates of this article. Their address is: 7630 Little River Turnpike, Annandale, VA 22003 - 2653)

Schriber, T., "Perspectives on Simulation Using GPSS" in Proc. of the 1988 Winter Simulation Conference, M. Abrams, ed., Pub. by The Soc. for Comp. Sim., San Diego, CA


Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 2
SAMPLE GPSS/H PROGRAMS

Running A GPSS/H Program

The best way to learn GPSS/H is to run numerous programs, each time you are introduced to a new topic. A GPSS/H program will have many different commands so it will not be until a few more topics are introduced that you will be able to write your programs yourself. However, it is possible to type up the programs and run them right away. For the present, do not worry what all the different commands are - each will be explained later.

What You Will Need

You will need a PC that has GPSS/H loaded. You must know how to create and edit files. The creation of files can be done using the DOS editor (probably the easiest), word processing software such as WordPerfect, MicroSoft Word, etc, that can create an ASCII file. The creation and editing of files will not be covered here.

The first problem we are going to solve is one where people enter a barber shop every 10 minutes. It takes the barber exactly 13 minutes to give a haircut. Obviously, this is a situation that will soon lead to the barber shop being overloaded with customers and so not realistic. But, it will introduce us to what the GPSS language looks like and what is the output from the program. The problem is to simulate the shop for 1 hour, starting at t = 0 when there are no customers in the shop. Figure 2.1 illustrates the time scale with customers arriving and leaving for the 60 minutes.

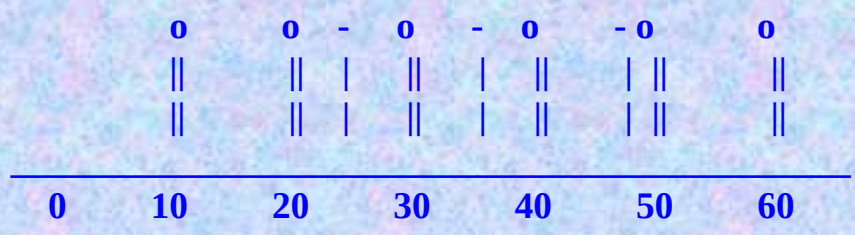


Figure 2.1 Representation of what is happening in the barber shop.

The barber will have nothing to do until t = 10 when the first customer arrives. The double lines at t = 10, 20, ... 60 represent the customers arriving at the shop. The single lines without the circles starting at t = 23 represent customers leaving. It should be easy for us to understand this system and be able to explain what is happening. Our first GPSS/H program will simulate this barber

shop. Although it is not necessary to follow the format given below for this exercise, try to type the program exactly as given. Imagine that you are typing the program on a numbered grid and each column has a position number as shown in Figure 2.2

```
                | |
- - - - - - - - |-| - - - - - - - --|-| - - - - - - - -
1 2 3 4 5 6 7 8 9 0|1| 2 3 4 5 6 7 8 9 0 1|2| 3 4 5 6 7 8 9 0
                | |             | |
                | |             | |
```

Figure 2.2 Line showing position numbers.

The GPSS/H commands are either blocks or statements (we will learn the difference later). Each line will have only one such command. The general form of a GPSS/H program block is

label operation operand(s) comment

The label goes in positions 2 - 9; the operation in positions 11 to 21, the operand(s) up to position 25 and the comments follow in any position as long as there is a blank space following the operand(s). It is important to remember that blanks are not permitted in labels, operations, or operands. Thus, if the block is to be:

```
GENERATE ,,,4 THIS A GENERATE BLOCK
```

it would be incorrect to have

```
GENERATE , , , 4 THIS IS A GENERATE BLOCK
```

It is possible to continue an operand from one line to another using the underscore, "_". If the underscore is placed after and code (but with no spaces!), the code is continued to the next line. For example,

```
ADVANCE 10,4
```

and

```
ADVANCE 10,_
    4
```

are identical. Once you use an underscore to continue a line, you can begin the continuation in any space up to position 25.

In the following program there are no labels so that the blocks or statements will begin in position 11. In some cases, there will be associated operands and these will begin in position 22. Also only capital letters are allowed in GPSS/H program lines. This restriction does not apply to comments.

The program will look as follows.

```
SIMULATE
GENERATE 10
QUEUE SEAT
SEIZE BARBER
DEPART SEAT
ADVANCE 13
RELEASE BARBER
TERMINATE
GENERATE 60
TERMINATE 1
START 1
END
```

If you have never seen a GPSS/H program, this must look strange but, like any programming language, this will become familiar to you with practice. Notice that there are no commands that correspond to input or output such as READ or WRITE. This is because you normally do not read data into a simulation program. But what about output? Here is where GPSS/H is so helpful. Whenever certain blocks appear in the program, there will automatically be output associated with that block. If you are studying a queueing situation as happens in our barber shop, output will automatically be produced as will be the case for other blocks that will be discussed later. If you want to have customized output, this is possible but will not be covered until quite a bit later.

The program you wrote must have the extension .GPS. Suppose the name of it is BARBER.GPS. To run it you need to be in the GPSSH directory and then type:

GPSSH BARBER NOXREF NODICT <cr>

The extension is not needed in the file BARBER.GPS. The commands NOXREF and NODICT are optional. They represent "no cross reference" and "no dictionary". If they are omitted there will be considerable output if the program has an error. Generally, you do not need this. These are optional and can be omitted. If you omit them, there will be additional output.

If your program was written with no errors, you will see a screen such as:

```
GPSS/H Release 3.0 12 May 1992 14:15:57
```

```
File: BARBER.gps

Compilation begins.

Pass 1 (with source listing)...

Pass 2...

Simulation begins.

GPSS/H IS A PROPRIETARY PRODUCT OF,
AND IS USED UNDER A LICENSE GRANTED BY,
WOLVERINE SOFTWARE CORPORATION
4115 ANNANDALE ROAD
ANNANDALE, VIRGINIA 22003-7500, USA

C:\GPSSH>
```

The return of the DOS prompt means that the program successfully ran. At the completion of the program, GPSS/H creates a list file which has the same name as the original file but now with the extension .LIS. To view the program you need to examine this file. This can be done using the same text editor used to create it (or simply by typing TYPE BARBER.LIS | MORE).

The output will look as follows:

```
GPSS/H 386 RELEASE 2.0 12 May 1992 14:41:47 FILE: BARBER.gps

LINE# STMT# IF DO BLOCK# *LOC OPERATION A,B,C,D,E,F,G COMMENTS

1    1      SIMULATE
2    2   1  GENERATE 10
3    3   2  QUEUE SEAT
4    4   3  SEIZE BARBER
5    5   4  DEPART SEAT
6    6   5  ADVANCE 13
7    7   6  RELEASE BARBER
8    8   7  TERMINATE
9    9   8  GENERATE 60
```

```
10    10   9   TERMINATE 1
11    11       START 1
12    12       END

STORAGE REQUIREMENTS (BYTES)
COMPILED CODE: 220
COMPILED DATA: 80
MISCELLANEOUS: 0
ENTITIES: 344
COMMON: 10000
-----------------------
TOTAL: 10644
Simulation begins.

RELATIVE CLOCK: 60.0000 ABSOLUTE CLOCK: 60.0000

BLOCK CURRENT TOTAL
1              5
2        1     5
3              4
4              4
5        1     4
6              3
7              3
8              1
9              1

--AVG-UTIL-DURING--
FACILITY   TOTAL    AVAIL    UNAVL    ENTRIES    AVERAGE
           TIME     TIME     TIME               TIME/XACT
BARBER     0.833                      4          12.500

           CURRENT    PERCENT    SEIZING    PREEMPTING
            STATUS     AVAIL      XACT       XACT
            AVAIL                   5

QUEUE      MAXIMUM    AVERAGE    TOTAL     ZERO
           CONTENTS   CONTENTS   ENTRIES   ENTRIES
SEAT          1         0.467      5         1
```

```
          PERCENT    AVERAGE     $AVERAGE    QTABLE    CURRENT
          ZEROS      TIME/UNIT   TIME/UNIT             CONTENTS
           34.1        5.665       8.596                  0

STATUS OF COMMON STORAGE

9424 BYTES AVAILABLE
576 IN USE
680 USED (MAX)

Simulation terminated. Absolute Clock: 60.0000

Total Block Executions: 30

Blocks / second: 30000

Microseconds / Block: 33.33

Elapsed Time Used (SEC)
PASS1: 0.05
PASS2: 0.06
LOAD/CTRL: 0.16
EXECUTION: 0.00
OUTPUT: 0.06
---------------------
TOTAL: 0.33

GPSS/H IS A PROPRIETARY PRODUCT OF, AND IS USED UNDER
A LICENSE GRANTED BY, THE WOLVERINE SOFTWARE CORPORATION,
4115 ANNANDALE ROAD, ANNANDALE, VIRGINIA 22003-2500, USA.
```

Let us now examine your output. It is going to look strange at first but you will soon become accustomed to interpreting the results. For the present we will simply ignore most of it.

The first line of interest to us will be the one that is:

```
RELATIVE CLOCK: 60.0000 ABSOLUTE CLOCK: 60.0000
```

This indicates that the simulation went for 60 simulated time units. Notice that there are two imaginary clocks in GPSS. Both start at time t = 0. We shall learn how to run a simulation for a certain period, stop execution and re-start it with most, but not all, of the statistics set back to zero. When this is done, the absolute clock would keep going but the relative clock is reset to zero each time the program is re-started.

Under MAXIMUM CONTENTS we see that there was a maximum of 1 person waiting. The TOTAL ENTRIES indicates that 5 people entered the queue. Even the first person who entered the barber shop and went immediately to the barber's chair is counted. This person is listed under the next heading ZERO ENTRIES as being 1. Of the 5 people who entered the shop 1 did not remain in the seat so the PERCENT ZEROS is 20.0. The average time for each person in the queue was 5.6. This is determined by noting that 5 people entered the queue. The first was there for 0 minutes, the second for 3 minutes, the third for 6 minutes, the fourth for 9 minutes and the fifth for 10 minutes, thus 28 divided by 5 is 5.6. The next entry is the total time in the queue but now divided by only those people who actually remained in the queue, namely 4. The CURRENT CONTENTS is 1 which indicates that one person is in the queue at the end of the simulation.

Example 2.1

Suppose that the arrival times and haircut times were reversed so that the customers arrive every 13 minutes and the barber can give a haircut in 10 minutes. The modifications necessary to do this simulation are as follows:

change the line

```
GENERATE 10
```

to

```
GENERATE 13
```

change the line

```
ADVANCE 13
```

to

```
ADVANCE 10
```

Make these changes and see if you can interpret the results.

A Look at the GPSS/H Code

We have just ran our first program and then some of the results were interpreted. You were told to be very careful about where and how the various commands were typed. This is because GPSS was first introduced when computers used punched cards exclusively for reading the program. This is no longer the case so the restrictions inherent with punched cards no longer apply. There are two ways to write your program, fixed and free format. Both will be presented here. We shall learn that in GPSS/H there are basically two types of program commands. One is called a statement and the other a block. The various properties of each will be discussed in subsequent Modules. The form of both are similar and so the discussion here will apply to both blocks and statements.

Fixed Format

In GPSS each separate line of the program will either be a statement or a block. (In a few rare situations the GPSS statement will be continued for two or more lines). A general format of a GPSS block consists of four separate items. These are:

1. **Label or location**
2. **Operation or block statement**
3. **Operands**
4. **Comments**

The form will always be:

label - operation - operand - comments

1. **The label starts in position 2 and goes through position 9. There is (normally) nothing in position 1 or 10.**
2. **Operations are in positions 11 through 20 with no spaces allowed.**
3. **Operands begin in position 25 (or before) and continue through to position 71. Spaces are NOT permitted in operands. (Keep this in mind later when you are learning how to do arithmetic operations as it is tempting to leave spaces around the plus or minus signs.) It is possible to continue to another line by putting the underscore character `_' in or to the left of position 72 in the Operand (not the Operation). The next line is read starting with the first non-blank character anywhere in positions 1-19.**
4. **A comment can be placed after a blank space after the operand. Most programs that have comments will have them generally all starting in the same column for ease of reading them. But be careful - not all statements have operands. For these, it is necessary to place comments starting in position 26 or anywhere after.**

Any line beginning with an asterisk is ignored.

5. Even though rule 3 specifies that operands are to begin in position 25 (or before), the programmer can specify where the operands are to begin by means of the OPERCOL statement. This has the form:

```
OPERCOL n
```

where n is the position the operands can begin. By default, this is 25. Thus,

```
OPERCOL 30
```

will tell the compiler to scan for the operands up to position 30. This can be very useful in certain situations such as when you have output that includes items that are nearly the same and you want them to line up underneath each other. It can also come in handy when you use nested DO loops.

The above may sound like a lot of "thou shall not's", but, in practice, the form of the GPSS/H lines of code are easy. The last item, about the OPERCOL, is rarely used if one sticks to fixed format.

The label must be either a letter or a number, but not more than 8 characters in length. It is possible to mix numbers and characters, providing the first character is a letter. Examples are:

```
JOEANNE
BILLYBUD
UPTOP
DOWN1
BACK1
UPTOP7
A123
```

but not

```
1JOE
B23$K
-1
J&B
```

To illustrate how the above works, take the program you just wrote and change it as follows.

```
***----------------------------***
```

```
* *
* MY FIRST GPSS PROGRAM *
* *
***----------------------------***
*
*
* I WONDER HOW LONG BEFORE IT STARTS TO
* MAKE SENSE
*
*------------------------------*
SIMULATE
*
COME GENERATE 10 THIS HAS TO DO WITH PEOPLE ARRIVING
QUEUE SEAT
SEIZE BARBER
DEPART SEAT
ADVANCE 13 I BET THIS HAS TO DO WITH
* GETTING A HAIRCUT-----------------------------------x
* THE NEXT LINE ILLUSTRATES SOMETHING RARE
RELEASE BA_
    RBER
* I HOPE THIS WILL WORK - IT LOOKS STRANGE
*
*
*
TERMINATE
GENERATE 60
TERMINATE 1
START 1
END
```

Except for the way the RELEASE block was written to illustrate how a continuation to the next line can be made, the above is the form most GPSS programmers follow. The initial comments are given using asterisks and describe the program while other comments are placed in the program by inserting them after a blank in the line after the operand. The comments here are all shown in capital letters but they could be in small letter as they are ignored by the compiler.

Notice that when you ran the program the second time you received a "warning.....". This is because the block

```
COME GENERATE 10
```

was never referred to. Do not be alarmed, this is just the way GPSS works. Once you give a block a

label, GPSS expects you to refer to it in the main program. It looks a bit messy but messages such as this can be a real help in debugging long programs.

Example 2.2

Even though we still have a ways to go in writing our programs, we can still use the one program we have written to learn a bit more about GPSS. From now on we will speak of the `processor' when we refer to how the program is being executed. To simulate the same shop for the barber working a bit faster, simply change the line:

```
ADVANCE 13
```

with

```
ADVANCE 12
```

Now the barber is cutting hair in 12 minutes (the shop will still overflow with people).

To simulate for 2 hours, change the line:

```
START 1
```

to

```
START 2
```

By now you should be able to understand the results of the second simulation.

Example 2.3

To simulate the barber cutting hair in 9 minutes change the line

```
ADVANCE 13
```

to

```
ADVANCE 9
```

Now there will not be a build up of customers. Run your program with this change for 2 hours of

simulated time and interpret the results.

Free Format

Since GPSS/H programs are now written on the screens of PC's, the restrictions that apply to fixed format have been relaxed. The way GPSS statements can now be typed is as follows:

1. **Labels must begin in either column 1 or 2. They can be up to 8 characters in length as can operands.**
2. **If no label is used, the operation portion can start in column 3 (or later).**
3. **Operands start following the operation. There need be only 1 blank between the operation and the operand. There may be more blanks, in which case the operand must begin in or before the position given by the OPERCOL.**
4. **Comments are placed in the program just as for fixed format.**
5. **Statements may be continued to another line as in fixed format.**

Thus, it is possible to return to our first program and type it as follows:

```
*-----------------------------
*
*
*
*-----------------------------
COME GENERATE 10
QUEUE SEAT
SEIZE BARBR
DEPART SEAT
ADVANCE 13
RELEASE BARBR
TERMINATE
GENERATE 6_
      0
TERMINATE
GENERATE 60
TERMINATE 1
START 1
END
```

There really is no advantage in writing programs as above. The programs presented in this book will all be written in fixed format for sake of uniformity. If you choose to write your programs in free format, this is perfectly acceptable.

The SIMULATE Statement

Every GPSS/H program must contain a SIMULATE statement. Although it need not be the first statement in the program, it usually is. Most programmers always start their programs with the SIMULATE statement. The general form of it is:

```
SIMULATE n
```

The operand n is optional. If used, it limits that amount of time in minutes which the program will run. This can be handy to use in the de-bugging stage to avoid infinite loops. For example,

```
SIMULATE 2.5
```

would limit the program to 2.5 minutes running time. It is possible to have the time limited to so many seconds if the letter S is placed after the SIMULATE operand:

```
SIMULATE 100S
```

would limit the execution time to 100 seconds. There is a caution with this statement. Be careful that you do not put a comment after it. If you do, the comment must start in or after position 25.

The END Statement

GPSS/H programs must have an END statement. This is simply the last statement in the program and acts as a directive to the compiler. Of all the many blocks and statements with GPSS, the SIMULATE and END should never cause you any problems.

Example 2.4

Let us do the same example of the barber shop again. (Soon we shall be doing other, more interesting problems.) In the last example you were told to run it for 2 hours of simulated time by changing the START 1 to START 2. The following program will simulate the barber shop for 10 customers having been given their haircuts. Write the program and run it.

```
SIMULATE
GENERATE 10
QUEUE SEAT
SEIZE BARBR
DEPART SEAT
```

```
ADVANCE 13
RELEASE BARBR
TERMINATE 1
START 2
END
```

After you run the program you should examine the output to see what the program results are. After a few more exercises you will understand the differences between the programs.

Exercises

1.Parts come along an assembly line with an interarrival rate of either 4, 5 or 6 minutes. A single worker takes either 4 or 5 minutes to work on each at station A. This worker can work on only one part at a time. Next parts go to station B where two identical workers take either 8, 9, or 10 minutes to do further work on the parts. Finally, an inspector takes exactly 5 minutes to check the parts.

Do a "hand simulation" for the first hour of operation starting at time 0. For the sake of being uniform with others who are doing this exercise assume that the various times by the worker are in exact order. Thus, the first worker will work on the first part in 4 minutes, the second part in 5 minutes, etc.

Your result should show the event, the time it takes place and a description of what is being done. The first part of your solution will look as follows:

event	time	description
1	4	first part comes along, next part will come at time 9.
2	4	first part will be worked on until time 8.
3	8	first part done at station A. Goes to station B.
4	8	first part is worked on at Station B until time 16
5	9	second part comes along, next part will come at time 13.

note: The GPSS/H program to do the simulation is:

```
SIMULATE
STORAGES (WORKERS),2
PARTS FUNCTION RN1,D3                PARTS COME ALONG
.333,4/.667,5/1,6
```

```
WORK1 FUNCTION RN1,D2                FIRST WORKER
.5,4/1,5
WORK2 FUNCTION RN1,D3                SECOND WORKERS
.333,8/.666,9/1,10
GENERATE FN(PARTS)                   PARTS COME
QUEUE WAIT1                          JOIN FIRST QUEUE
SEIZE FIRST                          FIRST WORKER IS USED
DEPART WAIT1                         LEAVE THE QUEUE
ADVANCE FN(WORK1)                    WORK ON FIRST PART
RELEASE FIRST                        FREE THE WORKER
QUEUE WAIT2                          JOIN SECOND QUEUE
ENTER WORKERS                        USE ONE OF THE WORKERS
DEPART WAIT2                         LEAVE THE QUEUE
ADVANCE FN(WORK2)
LEAVE WORKERS                        FREE THE WORKER
QUEUE WAIT3                          JOIN LAST QUEUE
SEIZE LAST                           USE THE WORKER
DEPART WAIT3                         LEAVE THE QUEUE
ADVANCE 5                            INSPECT PART
RELEASE LAST                         FREE THE INSPECTOR
TERMINATE                            PART DONE
GENERATE 60                          TIMER TRANSACTION
TERMINATE 1                          SIMULATION OVER
START 1
END
```

Run the above program and interpret the results. (as much as possible).

2. In order to run the above program for 8 hours (480 minutes) it is necessary to change the line of code:

GENERATE 60

to

GENERATE 480

Make this change and re-run the program.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 3
The GENERATE and TERMINATE Blocks
The START Statement
The GENERATE Block

The key to any simulation program is the GENERATE block. This block creates `transactions' that move through the program from program block to program block. These transactions can represent anything that you wish to model, such as ships entering a harbor, people coming to work, cars on a highway, a person entering a bank, etc. To give an idea of how a transaction is used, think of a person about to enter a barber shop with a single barber. The person is the transaction. He or she will enter the shop and let us suppose will do one of the following things.

1. **The barber is free so the person immediately sits in the chair and the haircut begins.**
2. **The barber is busy so the person will wait until he is free.**
3. **The shop has too many people waiting so the person will leave.**

We shall soon learn how to simulate the above situations with GPSS/H. A system to be studied will normally involve the passage of time and so an imaginary clock is used by the GPSS processor. This will be discussed next.

The Internal GPSS Clock

GPSS uses an internal clock that starts at zero when the program begins execution. The processor moves the clock forward in time as the program is executed.

These time units can represent seconds, minutes, hours, one tenth of a minute, etc., depending on what the programmer chooses them to represent. You will select the time unit to represent the problem being studied. In some cases a time unit, called the `basic time unit', of 1 second may be selected; in others, the basic time unit may be 1 hour. The clock in GPSS/H advances from event to event that are taking place in the simulation. For example, if some event is to take place at time t = 345.765 and the next one at t = 420.5, the internal clock will be advanced from where it is to time 345.765. It then will advance to time 420.5.

If a shop is being studied and time studies indicate that customers take 18 minutes to shop and 2 minutes to checkout, the basic time unit may be taken as 1 minutes. If the simulation is to run for 8 hours, then the 8 hours needs to be converted to the basic time unit or 480 time units. This basic time unit selected is generally obvious from the statement of the problem being modeled.

Creating Transactions

Transactions are created by the GENERATE block. It can have up to 9 operands and so can be quite involved. The simplest form of it is given by the following:

```
(label)  GENERATE  A
```

where A is either a positive integer or can be a variable (we will learn what forms these can have later). The operand A gives the times at which a transaction will be created:

The (label) is optional. We will learn when one uses one. For the present, it is not needed. For example,

```
(a) GENERATE 5
(b) GENERATE 100
```

In (a), a transaction is created every 5 time units, while in (b), a transaction is created every 100 time units. In both cases as long as the simulation is taking place, transactions will continue to be created every 5 or 100 time units.

In general, a transaction will be moved from block to block in a sequential manner, unless it encounters a block that transfers it elsewhere in the program. This non-sequential transfer only takes place where the programmer specifies it.

In most systems to be studied there is a degree of randomness involved. People do not enter a bank every 35 seconds, a barber does not take exactly 8.5 minutes for a haircut, etc. Thus, it is necessary to have randomness as a part of the simulation. One of the features of GPSS/H is that it is so easy to incorporate randomness into the program. One way to have randomness in the generation of transactions is given next.

The second form of the GENERATE block uses the B operand:

```
GENERATE  A,B
```

B can be a variable, but for the present it will be a positive integer. The above will generate a transaction over the interval A ± B, with each time having equal probability of happening.

```
GENERATE 8,2
```

means that a transaction is created every 8 ± 2 time units. This means that a transaction will be

created at an interarrival time from 6 to 10 with equal probability. The times are sampled from the interval (6.0000, 10.0000). The open brackets (a, b) indicate that the times of 6.0000 and 10.0000 are not included, but 6.0001 and 9.9999 are. This means that if the internal clock is at t = 1013.0000, and the GPSS processor sets t = 6.0000 as the time before the next transaction is created, the next transaction will enter the system at simulated time t = 1019.0000. We will learn later how to generate transactions that enter the system according to any statistical distribution.

Although it is not of concern how the processor works, you might be curious how the processor might generate the various times using random numbers. The processor has a built in random number generator that we will be referring to from time to time. Suppose you want to generate times from 10 to 24 with equal probability. Call these X. The random number will be called RN, where RN is a number from 0 to 1. Now consider the formula:

```
X = 10 + RN * (24 - 10)
```

Every time the random number is called up, a new value of X is obtained. As can be seen, a stream of times between 10 to 24 will be obtained. This is similar to the way that the GPSS/H processor works.

In using the GENERATE block you must be careful to avoid a block such as:

```
GENERATE 10,12
```

as this would eventually lead to an attempt to generate a transaction at a negative time, which is not allowed. However, it is possible to have:

```
GENERATE 10,10
```

Before going on with the GENERATE block, let us do another exercise. By now the GPSS/H program should be starting to make a bit of sense.

Example 3.1.

People arrive at a barber shop every 15 ± 6 minutes. The single barber takes 12 ± 4 minutes to cut hair. Simulate for 200 people having their hair cut. Assume that the barber works continuously, i. e., he does not leave the shop until 200 people have had their hair cut. Determine how busy the barber has been.

The program to do this is as follows:

```
SIMULATE
```

```
GENERATE    15,6      PEOPLE ENTER THE SHOP
QUEUE       SEAT      TAKE A SEAT
SEIZE       BARBER    IF BARBER FREE, BEGIN HAIRCUT
DEPART      SEAT      LEAVE THE SEAT
ADVANCE     12,4      RECEIVE HAIRCUT
RELEASE     BARBER    HAIRCUT OVER, BARBER IS FREE
TERMINATE   1         LEAVE THE SHOP
START       200       SIMULATE FOR 200 CUSTOMERS
END
```

Solution

Do not be too concerned at this time that you do not understand all the code given above. In fact, even interpreting the results will appear strange at this time. However, if you successfully run the program and look at the list file created by GPSS/H, you will see the following as a part of the output:

```
RELATIVE CLOCK: 3005.4781 ABSOLUTE CLOCK: 3005.4781

--AVG-UTIL-DURING--
FACILITY TOTAL ENTRIES
BARBER 0.804 200
```

The above is interpreted as follows: The barber worked for approximately 3005 minutes or 50 hours straight to take care of the 200 customers. He was busy 80.4% of the time. This result is to be expected. Customers arrive at random but with an interarrival time of 15 minutes. The barber takes an average of 12 minutes to give a haircut. Hence, in the long run, one would expect the barber to be busy for 12/15 or 80% of the time.

When the 200th customer was done, the barber closed shop and left, even though there may have been other customers in the shop waiting. Later, when we learn more about the GPSS/H language, it will be possible to write the code to model a more realistic examples.

Example 3.2.

Suppose the barber decides to buy special, fancy new equipment that can really speed up his hair cutting. He can now cut hair in 10 ± 3 minutes. Re-run Example 3.1 and examine the results.

This is done by changing the operands of the ADVANCE block so that it now is

```
ADVANCE 10,3.
```

More General Cases of the GENERATE Block

There are other forms of the GENERATE block. The ones we will use at the present are:

```
GENERATE A,B,C,D,E
```

The C, D and E operands do the following:

C is called the offset time for the first transaction. No transaction will enter the system until this time.
D is the maximum number of transactions to be generated.
E is called the priority. This can be very useful.

When the E operand is used, the transactions are given a priority level specified by it. Often this operand is omitted and so the priority level is, by default, 0. The priority levels are integers which can be from -2,147,483,632 to +2,147,483,632. In practice only a few priorities are needed and so one normally uses priorities such a 1, 2, 5, 10, etc.

If one transaction has a higher priority than another, it is placed ahead of it in queues as well as given preferential service in case of a time tie between transactions. In most queueing systems, the service criteria is known as `First in-First out.' (FIFO). This means that if the first person to arrive for service has to wait, he or she will be served before later arrivals. A typical example to illustrate the case of a time tie is the situation when a car arrives at a petrol station. Suppose that there is only room for 6 cars total and, if there are 6 cars at the station, an arriving car will leave. A time tie occurs when a car arrives at exactly the same time as one is through being serviced, the arriving one will not leave if it has a higher priority than the one that is about to leave. This, of course, is what will happen in a real life situation. We will examine this concept of priority later.

Several examples of the GENERATE block are considered next.

```
(a) GENERATE 15,3,100,3,1
(b) GENERATE 100,3,200,400,3
(c) GENERATE 20,4,500,7,8
(d) GENERATE 30,0,0
```

In (a), a transaction is created every 15 ± 3 time units. The first does not enter the system until t = 100. Only 3 of these transactions will be created and each will have a priority level of 1.

In (b), a transaction is created every 100 ± 3 time units. The first enters the system at t = 200

and only 400 of these transactions will be created, each having a priority of 3.

In (c), transactions are created every 20 ± 4 time units. The first enters the system at t = 500. Only 7 such transactions are created, each with priority 8.

In (d), transactions are created every 30 time units. The first will enter the system at time 0. Contrast this with the block:

```
GENERATE 30
```

Here the first transaction will enter the system at time 30, not at time 0. If you do not wish to use all the operands, you put commas in their place - not blanks, i.e.

```
GENERATE 100,,,1
```

will generate a single transaction at t = 100.

```
GENERATE 100,,,,1
```

will generate a transaction every 100 time units each having a priority level 1.

```
GENERATE ,,,5
```

will generate 5 transactions immediately. This may seem a strange thing to do but this will be very important in many of our simulations. The above is the same as:

```
GENERATE 0,0,0,5
```

If you want to study a program that will have 6 ships sailing from one port to another you might start the simulation off by using the block:

```
GENERATE ,,,6
```

This puts 6 ships into the system at time t = 0.

Example 3.3

Go back to Example 3.1. Now suppose that when the barber arrives at work he finds 3 people waiting for haircuts. Change the program to reflect this. The program for this is:

```
    SIMULATE
    GENERATE    ,,,3        3 PEOPLE WAITING
    TRANSFER    ,DOWN       SEND THEM TO THE DOOR
    GENERATE    15,6        PEOPLE ENTER THE SHOP
DOWN QUEUE      SEAT        TAKE A SEAT
    SEIZE       BARBER      IF BARBER FREE, BEGIN HAIRCUT
    DEPART      SEAT        LEAVE THE SEAT
    ADVANCE     12,4        RECEIVE HAIRCUT
    RELEASE     BARBER      HAIRCUT OVER, BARBER IS FREE
    TERMINATE   1           LEAVE THE SHOP
    START       200         SIMULATE FOR 200 CUSTOMERS
    END
```

Example 3.4

Suppose that the customers in Example 3.3 arrive as before, but now it is desired to have them start arriving at 1 minute after the barber shop opens. The only change in the previous program is that the first GENERATE block is now

```
    GENERATE 15,6,1 PEOPLE ENTER THE SHOP
```

Incorporate this change and compare the output. It is possible to have more than one GENERATE block in a program and, in fact, this is nearly always the case. The only caution is that a transaction can never enter a GENERATE block. Thus one could have a program such as:

```
    GENERATE    1
    --------
    --------
    --------
    --------
    TERMINATE
    GENERATE    5000
    TERMINATE   1
```

The next section will explain what the TERMINATE block does.

The TERMINATE Block

Transactions enter the system by means of the GENERATE block. Eventually, in most

simulations, the transactions will have to leave the system. This is done by means of a block known as the TERMINATE block. It is quite simple in form:

```
TERMINATE n
```

where n is a positive integer, including 0. If n is omitted, as it often is, it is taken to be 0. Every time a transaction enters this block, it is immediately removed from the system. Some examples of it are:

```
TERMINATE  1
TERMINATE  20
TERMINATE  5
TERMINATE  0
```

This operand n in TERMINATE n has nothing to do with the transaction being removed from the system. Only one transaction at a time enters the TERMINATE block. The TERMINATE block always removes this one transaction. As we shall see, this operand is used to control the execution of the program in connection with another statement.

The START Statement

Every GPSS/H program must have a START statement. The simplest form of it is:

```
START  n
```

where n must be a non-zero positive integer. Some examples are:

```
START 1
START 10
START 200
START 66
```

The number, n, is a counter for controlling the running of the program. While the program is being executed, the counter is being decremented. When it becomes zero or negative, the program stops execution. The GPSS/H processor then creates a file, name.LIS, where name is the name of the original GPSS/H file. This file contains the results of the simulation and can be viewed using the same text editor used to create the original GPSS/H file.

The way the processor knows that the program is finished is as follows:

1. The counter, n, is set aside.
2. Whenever a transaction goes through a TERMINATE block that has an operand, this operand value is subtracted from n.
3. When n becomes 0 or negative, the simulation is finished and the report produced.

For example assume that the GPSS/H programs contain only the following TERMINATE and START lines of code:

```
a) TERMINATE 2
   ..........
   START 10
b) TERMINATE
   .........
   TERMINATE 3
   .........
   START 13
c) TERMINATE 4
   .........
   START 1
```

In a), the program will execute until 5 transactions have passed through the TERMINATE 2 block.

In b), the program will run until 5 transactions have passed through the TERMINATE 3 block. Any transaction that passes through the first TERMINATE block will have no effect on the execution time of the program.

In c), the program will run until 1 transaction has passed through the TERMINATE 4 block.

Most of the programs run so far have had blocks such as the following:

```
GENERATE    480
TERMINATE   1
START       1
```

The effect of the above is to put a single transaction into the system at time 480. This transaction is immediately removed via the TERMINATE 1 block. The 1 in its operand causes the counter of 1 which was given by the START 1 statement to be decremented to 0. Thus, the program stops execution at time 480.0000. Transactions that have no other effect on the program other than to stop the execution are called timer transactions.

A GPSS/H program starts execution when the first START statement is encountered. If there are more statements after this first START statement, they are initially ignored. When the program is through with execution as specified by the first START statement, whatever commands are given after it are then done.

During the compiling stage, transactions are primed to move through the system at times given by the GENERATE block. The transactions are placed on a time axis called the Current Events Chain (CEC). When the program begins execution the Processor takes over and moves the transactions one at a time as far as the transaction can move. After it moves a transaction, it goes back to the CEC and moves the next transaction to be moved. This continues as long as the program is running. The first transaction to be moved is the one positioned on the CEC at the earliest time, the next is the one at the next earliest time, etc. The transaction is moved from block to block in a sequential manner, unless the program specifies other wise. A transaction is moved until one of the following things happen:

1. The transaction is removed from the system. This is done via the TERMINATE block.
2. The transaction is put on another chain. These will be covered later.
3. The transaction is blocked and cannot enter a sequential block.

Exercises 3.1

1. What will happen when the following GENERATE blocks are used in a program:

```
a) GENERATE 100,30,,5
b) GENERATE ,,1000,4
c) GENERATE 1000,200,20,100
d) GENERATE 500,400,1
e) GENERATE ,,,4
```

2. You are observing trucks at point A. Every 4 minutes one truck passes this point. The trucks travel along a road for 5 minutes and then leave the system. In 20 minutes how many trucks have you observed? Assume that the first truck does not pass you until 4 minutes have passed.

The GPSS/H program to simulate this is:

```
SIMULATE
GENERATE   4
ADVANCE    5
TERMINATE  0
GENERATE   20
```

```
TERMINATE  1
START      1
END
```

3. In 2., suppose the trucks pass point A every 4 minutes but starting at t = 0. Change the appropriate line in your program.

4. In 2., a truck will pass point A every 4 ± 2.5 minutes. Change the appropriate line in your program.

5. In 2., you may have noticed that after 20 minutes the computer program said that you observed only 4 trucks. What happened to the truck at time 20? Suppose you change the program to give the trucks you are observing a Priority of 1. What line needs to be changed? How does this effect the program results?

6. In 2, simulate for 40 minutes.

7. Write the GPSS GENERATE block to:

a) have transactions enter the system every 5 time units.
b) have transactions enter the system every 100.6 time units.
c) have transactions enter the system every 10 ± 6.5 time units.
d) have transactions enter the system every 7 time units starting at time 100.
e) have transactions enter the system every 120 ± 35.6 time units beginning at time 200.
f) have 5 transactions enter the system at time 0.
g) have transactions enter the system every 100 time units beginning at time 80 and only 10 enter the system from this block.
h) have only 3 transactions enter the system at time 500. These transactions have priority 5.
i) have transactions with priority 5 enter the system every 6 ± 3.4 time units, staring at time 400. Only 6 of these are to enter the system.

8. What would happen if you had the block:

```
GENERATE 120,130
```

9. What would happen if you had the block:

```
GENERATE 4,1,,,-1
```

10. What do the following lines of code do:

```
GENERATE ,,480,1
TERMINATE 1
START 1
```

11. The following code is used to time the running of a GPSS program. What time will the simulated clock show at the completion of the program?

```
a) GENERATE   480
   TERMINATE  2
   START      2
b) GENERATE   4800
   TERMINATE  5
   START      10
c) GENERATE   1000
   TERMINATE  3
   START      7
d) GENERATE   200
   TERMINATE  10
   START      1
```

12. What will the simulated clock read when the following program is done running:

```
SIMULATE
GENERATE   100
TERMINATE  1
GENERATE   150
TERMINATE  2
START      10
END
```

13. Consider the following code:

```
GENERATE   100
--------
--------
TERMINATE  1
GENERATE   150
--------
--------
TERMINATE  2
START      10
```

Assuming no other TERMINATE blocks, for how long will the program run. Show by means of a table.

Solutions

1. a) Transactions are created every 100 ± 30 time units. Only 5 of these will enter the system.
   b) 4 transactions are created. They will enter the system at time 1000.
   c) Transactions are created every 1000 ± 200 time units. They will start to enter the system at time t = 20. 100 of these transactions will enter the system.
   d) Transactions are created every 500 ± 400 time units. The first enters the system at time t = 1.
   e) 4 transactions are created at time t = 0.000.

2. You will observe 4 trucks pass you. At time t = 20, the 5th truck is in front of you.

3. Change Line 2 to

```
GENERATE 4,,0
```

4. Change Line 2 to

```
GENERATE 4,2.5
```

5. Change Line 2 to

```
GENERATE 4,,,,1
```

The 5th truck will now have passed by you before the program ends.

6. This can be done in several ways. Some of them are:

```
a) GENERATE   40   Line 5
b) GENERATE   20   Line 5
   START      2    Line 7
c) GENERATE   10   Line 5
   START      4    Line 7
```

```
7.  a) GENERATE   5
    b) GENERATE   100.6
    c) GENERATE   10,6.5
    d) GENERATE   7,,100
    e) GENERATE   120,35.6,200
```

```
  f) GENERATE   ,,,5
or   GENERATE   0,0,0,5  (the first is the commonly used method)
  g) GENERATE   100,,80,10
  h) GENERATE   ,,500,3,5
  i) GENERATE   6,3.4,400,6,5
```

8. You cannot go backwards in time. Thus, the spread of 120 ± 130 would be from -10 to 240. An attempt to generate a transaction at a negative time would eventually occur (depending on the random numbers used). This would lead to an error.

9. A transaction is created every 4 ± 1 time unit with priority -1. (Only GPSS/H allows negative priorities).

10. The first transaction enters the system at time 480 according to the C Operand. The D operand indicates that only 1 such transaction would enter the system. You may note that this is equivalent to:

```
GENERATE    480
TERMINATE   1
START       1
```

If the aim of the analyst is to run the simulation for 480 time units, the effect of the different lines of code is exactly the same.

11. a) The internal GPSS counter is initially set at 2 because of the START 2 statement. At time 480 a transaction enters the system and is immediately terminated. The TERMINATE operand is 2 so the counter is decremented by this amount. Since this sets it to zero, the program stops at this time, namely, t = 480.0000

b) t = 9600.0000. At time 4800.0000 the counter is decremented from 10 to 5. At time 9600.0000 the counter goes to 0.0000.

c) At time 1000.0000 the counter is decremented by 3 to 4. At time 2000.0000 the counter is decremented to 1. At time 3000.0000 the counter is decremented to -2. Since it is now less than zero execution will stop.

d) At time 200.0000 the counter is decremented by 10 to -9. This stops execution of the program.

13. Timer transactions enter the system at times t = 100.0000, 150.0000, 200.0000, 300.0000, 400.0000, 450.0000, 500.0000, etc. The counter is decremented by 1 when the transaction from

```
GENERATE 100
```

enters the system and by 2 when the transaction from

```
GENERATE 150
```

enters the system. This is shown as follows:

time	counter	decrement amount
0	**10**	**0**
100	**9**	**1**
150	**7**	**2**
200	**6**	**1**
300	**3**	**3**
400	**2**	**1**
450	**0**	**2**

Thus, the clock will read 450.0000

Exercises 3.2

A small mine has a single shovel which can load only 1 truck at a time. Loaded trucks travel to a single dump area where they dump and then return to the shovel. More than one truck can dump at a single time. Relevant times are:

operation	time (in min.)
load a truck	3.2 ± 1.1
travel to dump	5.3 ± 1.7
dump	1.2 ± 1.3
return	4.1 ± .8

Determine a plot of loads dumped/8hr shift vs. no of trucks in the mine for 2 through 8 trucks. Simulate for 20 shifts of 8 hours each.

Solution

The program to solve the problem is as follows:

```
      SIMULATE
      INTEGER     &I
TRUCK GENERATE    ,,,&I
UPTOP QUEUE       WAIT
      SEIZE       SHOVEL
      DEPART      WAIT
      ADVANCE     3.2,1.1
      RELEASE     SHOVEL
      ADVANCE     5.3,1.7
      ADVANCE     1.2,.3
      ADVANCE     4.1,.8
      TRANSFER    ,UPTOP
      GENERATE    480*20
      TERMINATE   1
      DO   &I=2,8
      CLEAR
TRUCK GENERATE    ,,,&I
      START       1
      ENDDO
      END
```

Create the above text file naming it FIRST.GPS (the name can be different but recall that the extension must be GPS) using a text editor such as EDLIN or MS DOS 5's EDIT. You can also use Wordperfect to create a DOS file. Copy it to the GPSS/H directory and then,

GPSSH FIRST NOXREF NODICT <cr>

If the program runs successfully, you will see the DOS prompt. To view the results,

TYPE FIRST.LIS | MORE

Alternately, you can use your test editor to view the results.

1. Write the GPSS code to:

   a) have transactions enter the system every 5 time units.
   b) have transactions enter the system every 100.6 time units.
   c) have transactions enter the system every 10 ± 6.5 time units.

d) have transactions enter the system every 7 time units starting at time 100.
e) have transactions enter the system every 120 ± 35.6 time units beginning at time 200.
f) have 5 transactions enter the system at time 0.
g) have transactions enter the system every 100 time units beginning at time 80 and only 10 enter the system from this block.
h) have only 3 transactions enter the system at time 500. These transactions have priority 5.
i) have transactions with priority 5 enter the system every 6 ± 3.4 time units, staring at time 400. Only 6 of these are to enter the system.

2. What would happen if you had the block:

```
GENERATE 120,130
```

3. What would happen if you had the block:

```
GENERATE 4,1,,,-1
```

4. What do the following lines of code do:

```
GENERATE     ,,480,1
TERMINATE    1
START        1
```

5. Write a GPSS program to determine the number of trucks to have in a mine if the mine is a simple one shovel/load/haul/dump operation. Relevant times are:

load: 4 ± 1.2 minutes
haul: mean 5.6, std dev 1.1 normally dist
dump: mean 1.2, std dev .2 " "
return: mean 4.5, std dev .75 " "

Only one truck can be loaded at a time. There is no such restriction on dumping.

Solutions THE GENERATE BLOCK

1. **a) GENERATE 5**
   b) GENERATE 100.6
   c) GENERATE 10,6.5
   d) GENERATE 7,,100
   e) GENERATE 120,35.6,200

```
    f) GENERATE ,,,5
  or   GENERATE 0,0,0,5 (the first is the commonly used method)
    g) GENERATE 100,,80,10
    h) GENERATE ,,500,3,5
    i) GENERATE 6,3.4,400,6,5
```

2. You cannot go backwards in time. Thus, the spread of 120 +/- 130 would be from -10 to 240. An attempt to generate a transaction at a negative time would eventually occur (depending on the random numbers used). This would lead to an error.

3. Transactions have priorities from 0 to 127 with the lower being the most important (this seems contrary to the way we think of priority, but that is one of the peculiarities of the way GPSS works). A attempt to give a transaction a priority of -1 would result in an error.

4. The first transaction enters the system at time 480 according to the C Operand. The D operand indicates that only 1 such transaction would enter the system. You may note that this is equivalent to the more commonly used code of:

```
    GENERATE   480
    TERMINATE  1
```

If the aim of the analysist is to run the simulation for 480 time units, the effect of the different line of code is exactly the same.

5. The code to solve the problem is:

```
     SIMULATE                    COMPILER DIRECTIVE
     GENERATE   ,,,4             PUT 4 TRUCKS IN THE MINE
UPTOP QUEUE     HALT             TRUCKS JOIN QUEUE AT SHOVEL
     SEIZE      SHOVEL           ATTEMPT TO USE THE SHOVEL
     DEPART     HALT             LEAVE THE QUEUE
     ADVANCE    4,1.2            LOAD A TRUCK
     RELEASE    SHOVEL           FREE THE SHOVEL
     ADVANCE    RVNORM(1,5.6,1.1)  TRAVEL TO DUMP
     ADVANCE    RVNORM(1,1.2,.2)   DUMP
     ADVANCE    RVNORM(1,4.5,.75)  RETURN TO SHOVEL
     TRANSFER   ,UPTOP           RETURN TO THE QUEUE
     GENERATE   480*20           TIMER TRANSACTION
     TERMINATE  1                REMOVE TIMER TRANSACTION
     START      1                BEGIN SIMULATION
```

```
END                                    END COMPILATION
```

The program simulates for 20 days or 20 shifts of 480 minutes. Selected output from running the program is as follows:

number trucks	loads dumped	shovel util.	average queue
2	1238	.516	.027
3	1815	.756	.115
4	2265	.939	.383
5	2401	1.000	1.183
6	2411	1.000	2.160

Thus, the optimum number of trucks to have is 5.

Solutions

RUNNING THE PROGRAM

1. a) The internal GPSS counter is initially set at 2 because of the START 2 statement. At time 480 a transaction enters the system and is immediately terminated. The TERMINATE operand is 2 so the counter is decremented by this amount. Since this sets it to zero, the program stops at this time, namely, t = 480.0000

b) t = 9600.0000. At time 4800.0000 the counter is decremented from 10 to 5. At time 9600.0000 the counter goes to 0.0000.

c) At time 1000.0000 the counter is decremented by 3 to 4. At time 2000.0000 the counter is decremented to 1. At time 3000.0000 the counter is decremented to -2. Since it is now less than zero the program will stop.

d) At time 200.0000 the counter is decremented by 10 to -9. This stops execution of the program.

2.
```
GENERATE    240
TERMINATE   1
START       2
```

3. Timer transactions enter the system at times t = 100.0000, 150.0000, 200.0000, 300.0000, 400.0000, 450.0000, 500.0000, etc. The counter is decremented by 1 when the transaction from GENERATE 100 enters the system and by 2 when the transaction from GENERATE 150 enters

the system. This is shown as follows:

time	counter	decrement amount
0	**10**	**0**
100	**9**	**1**
150	**7**	**2**
200	**6**	**1**
300	**3**	**3**
400	**2**	**1**
450	**0**	**2**

Thus, the clock will read 450.0000

4. a) set A = 0 and B = 1
   b) set A = 1 and B = 0
   c) set A = 1 and B = 1

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 4
THE TRANSFER BLOCK: PART I

A transaction will normally move from block to block in a sequential manner. When the processor moves it along the current events chain, the transaction moves as far as it can move. As we learned in Chapter 3, eventually, one of three things happens to it to cause the processor to re-scan the CEC for the next transaction to be moved. The possible things that can happen to the transaction are:

a) it is terminated
b) it is put on some other chain
c) it is blocked and denied access to the next block.

It is possible to have the transaction move in a non-sequential manner. This is the purpose of the TRANSFER block. The TRANSFER block acts much like the GO TO statement found in other programming languages such as Fortran. However, in GPSS, there are several forms of it that can be used. In this chapter, the three most common ones will be discussed.

Case I. The unconditional TRANSFER block.

The form of this is:

```
TRANSFER ,(label)
```

The (label) is the label of a block. Examples are

```
TRANSFER ,DOWN
TRANSFER ,WAIT
TRANSFER ,UPTOP
```

The comma before the block label where the transaction is to be routed is essential.

When a transaction enters the TRANSFER block it is immediately sent to the block with the label given by its operand. If the block will not admit the transaction, it is held in the TRANSFER block until a later scan of the CEC by the processor. The TRANSFER block always admits transactions. It is possible to have

```
TRANSFER ,n
```

where n is a non-zero positive integer. In this case, the transaction is transferred to the block number n, counting from the top. Thus,

```
TRANSFER ,6
```

will send the transaction to the 6th block in the program. This can be a bit confusing in the case of large programs and is not as easy to follow as when block labels are used. Also, if a program is changed afterwards by adding additional blocks, all such TRANSFER blocks have to be changed. For these reasons, this form of the TRANSFER block will not be used in hthis chapter.

Case II. The conditional TRANSFER block.

The form of this is:

```
TRANSFER .xyz,block1,block2
```

where xyz is a decimal of no more than three digits (it can be less), $block_1$ is a block label as is $block_2$. The label, $block_1$, is optional but not the last label.

Examples of this are:

```
TRANSFER .123,DOWN,UPTOP
TRANSFER .5,,AWAY
TRANSFER .007,NEXT1,NEXT2
TRANSFER 0.9,UPTOP,DOWNX
```

Here is how this version of the TRANSFER block works. The transaction is sent to the block with the label $block_2$ a fraction of times. This fraction is given by the decimal in the first operand position. The rest of the time the transaction is routed to the block with the label $block_1$. If this label is missing (as it mostly is), the transaction is routed to the next sequential block. Some examples of this block are:

```
TRANSFER .333,DOWN,OUT
```

33% of the time the transaction is sent to the block with the label OUT. The remainder of 67% of the time the transaction is sent to the block with the label DOWN.

```
TRANSFER .8,,AWAY
```

80% of the time the transaction is sent to the block with the label AWAY. The remainder of the time the transaction continues to the next sequential block.

Exercises 4.1

1. An art gallery has 2 show rooms A and B. People arrive every 45 ± 20 seconds. 40% plan to visit only gallery A and then leave. The rest go to galley B first. It takes 220 ± 80 seconds to visit galley A and 300 ± 60 second to visit gallery B. Of the people who visit galley B 35% leave but the rest then go to galley A. (Strangely enough, no one ever goes to gallery A first and then gallery B). Simulate for 4 hours.

2. Change Exercise 1 to simulate for 200 people who visit room B and then leave the shop without visiting room A.

3. People come to an escalator (actually, a moving sidewalk) at an airport. They arrive at a rate of 35 ± 15 seconds. 30% choose to walk along side of it. Of the people who use the moving sidewalk, 25% walk/run on it. The escalator moves people who do not walk in exactly 55 seconds. Those who walk on it take 40 ± 12 seconds. Those who walk along side of it take 60 ± 32 seconds. Simulate for an hour.

4. Change Exercise 3 to simulate for 100 people who use the escalator. How long does this take?

5. Refer to the Figure 4.1 which gives a diagram of traffic flow. All streets are one way traffic.

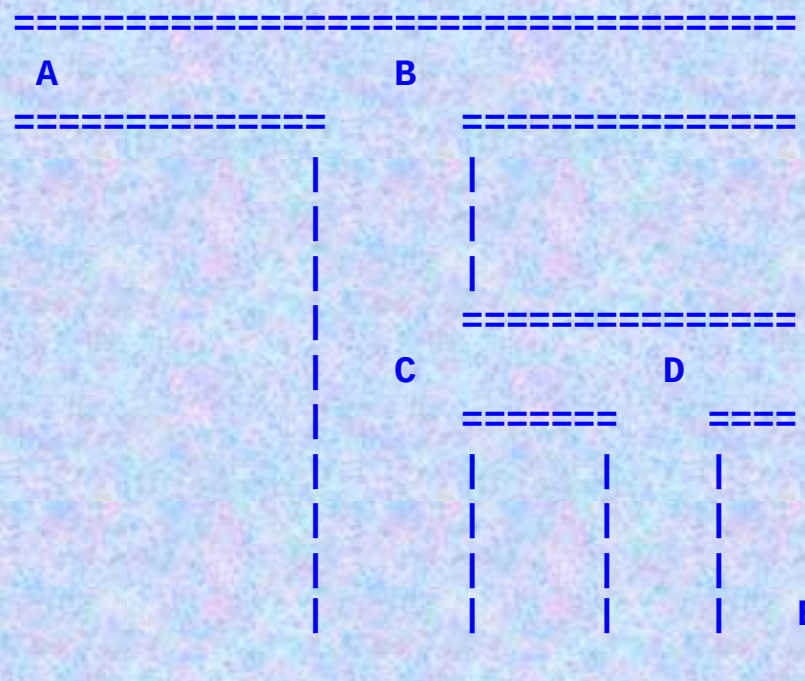


Figure 4.1 Sketch of traffic flow

Cars enter at A every 10 ± 3.2 seconds
They take 20 ± 8 seconds to travel to B
At B 30% go straight, the rest travel to C
It takes 15 ± 6 seconds to travel to C
At C 26% go straight, the rest turn to D
It takes 14 ± 3.1 seconds to travel from C to D
At D 31% of them turn to E (where you live), the rest go straight
It takes 12 ± 3 seconds for cars to travel from D to E.

You are going to protest that too many cars go by your house in an 8 hour period. Determine this number. (Note: ignore the fact that traffic patterns vary during the day).

Case III

The following form of the TRANSFER block is not as common as the first two but can be extremely helpful. It is called the TRANSFER BOTH mode. The general form of it is:

TRANSFER BOTH,$label_1$,$label_2$

The word BOTH must be in operand A position. The way the block works is as follows:

A transaction enters the block and attempts to go to the block given by $label_1$. If it can, it proceeds there. If not, it attempts to enter the block given by $label_2$. If it can, it does so. If it cannot enter either block, it remains in the TRANSFER BOTH block until a new scan of the CEC takes place. If the first label is missing, the next sequential block is taken. In fact, the most common form of the TRANSFER BOTH block is:

TRANSFER BOTH,,AWAY

where AWAY refers to a block that will always accept the transaction. Thus, the transaction "looks ahead" to the next sequential block and attempts to enter it. If it cannot, it is transferred to the block labeled AWAY. The next exercise illustrates a very nice application of the block.

Example 4.1

People arrive at a shop every 8 ± 3.5 minutes. The shop has five servers who work at the rate of 40 ± 15 minutes. There is a single chair to wait in case all of the servers are busy. If the shop is full, i. e., all five servers are busy and a person is in the chair, arriving customers leave and do not return. Simulate for 20 days operation (480 minutes per day) and determine how many customers are turned away. Even though you do not have enough GPSS to follow all the lines of code, it will be instructive to run the program.

Solution

The program to solve this Example is as follows:

```
      SIMULATE
      STORAGE    S(SEAT),1/S(WORKER),5
      GENERATE   8,3.5
      TRANSFER   BOTH,,AWAY
      ENTER      SEAT
      ENTER      WORKER
      LEAVE      SEAT
      ADVANCE    40,15
      LEAVE      WORKER
      TERMINATE
AWAY  TERMINATE
      GENERATE   9600
      TERMINATE  1
      START      1
      END
```

note: to run for two seats you change the line of code:

```
      STORAGE S(SEAT),1/S(WORKER),5
```

to

```
      STORAGE S(SEAT),2/S(WORKER),5
```

The following exercises are taken from Gordon, 1975.

Exercises

6. People arrive at a newsstand at the rate of one every 10 ± 5 seconds. Most people buy only one paper but 20 percent buy two papers. It takes 5 ± 3 seconds to buy one paper and 7 ± 3 seconds to buy two papers. Simulate the sale of 100 papers, starting from the time the newsstand opens.

7. In Exercise 6, suppose that every morning the newsstand always has two people waiting to purchase a newspaper. Add the code to include this.

8. A series of moving stairways carry customers in an upward direction between four floors of a department store. People arrive at the foot of the stairs, on the first floor, at the rate of one every second. Some people walk on the stairs. As a result, the time to transfer between any two floors is found to be 20 ± 10 seconds. The destinations of the customers are as follows: second floor, 50 percent; third floor, 25 percent; and fourth floor, 25 percent. Simulate the arrival of 100 people on the top floor, starting from the time the store opens.

9. Twenty people simultaneously take a test that requires 5 ± 2 minutes. Their chance of success is such that 20 percent pass on each trial. Those that fail wait 10 minutes before taking the test again (it still takes them 5 ± 2 minutes when they re-take it). They keep retrying until they finally pass. How long does it take for everyone to pass?

10. Cars bring spectators to a sports event at the rate of one car every 20 ± 10 seconds. The percentages of cars with a given number of passengers are as follows: 1 passenger, 10 percent; 2 passengers, 30 percent; 3 passengers, 45 percent; and 4 passengers, 15 percent. Find how long it takes for 1000 people to arrive.

11. People arrive at a cafeteria at the rate of one every 15 ± 5 seconds. There are two counters, A and B, and people want items from them in the following proportions: A only, 30 percent; A and B, 60 percent; and B only 10 percent. Simulate the arrival at the cafeteria of 100 people.

12. The delivery of some product is being limited by the availability of suitable containers. New containers are being made at the rate of one every 20 ± 5 minutes. They are filled and dispatched as soon as they are ready. Delivery takes 40 ± 10 minutes. About one in every 50 containers is damaged beyond repair during delivery. The rest are returned, taking 40 ± 10 minutes, and are immediately reused for another delivery. Beginning from time zero, find how many containers will be in the process of delivery after 8 hours.

13. A subway station has two entrances. Passengers arrive at entrance 1 at the rate of one every 10 ± 5 seconds, and they move along a corridor that takes 15 ± 5 seconds to walk. At entrance 2, passengers arrive at the rate of one every 5 ± 2 seconds and they walk along a corridor that takes 20 ± 8 seconds. The two streams of passengers merge to pass along a third corridor for 5 ± 3 seconds. At the end of that corridor, 60 percent of the passengers turn for the northbound platform, the rest turn for the southbound platform. Simulate the arrival of the first 100 passengers on the southbound platform, starting with an empty system.

14. Parts that are manufactured at the rate of one every 50 ± 10 seconds go through an inspection that takes 30 ± 10 seconds. The inspection passes 85 percent of the parts. Of the remainder, 5 percent are scrapped, and the rest are sent for reworking. Reworking takes 100 ± 30 seconds, after which the parts are again sent for inspection with the same probability of rejection. Simulate the acceptance of 100 parts. How many parts have been reworked by that time?

15. The following program contains some GPSS/H lines of code that will be covered in later chapters. These two lines are the ones with the comments. What the does is to generate 1000 transactions and transfer them to one of ten TERMINATE statements at random. The program is:

```
    SIMULATE
    RMULT      12345 RANDOM NUMBER SEED
    GENERATE   ,,,1000
    TRANSFER   ,FRN1*10+3 RANDOM TRANSFER
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    TERMINATE  1
    START      1000
    END
```

A portion of the output is:

```
RELATIVE CLOCK: 0. ABSOLUTE CLOCK: 0.

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               1000   11              105
2               1000   12              95
3               110
4               101
5               114
6               104
7               84
8               93
```

```
9               98
10              96
```

The expected number of times each of the TERMINATE blocks was entered by a transaction was 100. This was not always the case, with the maximum deviation being 16 for the TERMINATE block that was entered only 84 times. The line of code:

```
      RMULT   12345
```

is used to position the random number generator. Re-run the program with a different random number seed by changing the value of the operand to some other number. Then, re-run the program for 10000 transactions and see how close to the expected number of entries there were.

Solutions

1.
```
      SIMULATE
      GENERATE    45,20          PEOPLE ARRIVE
      TRANSFER    .40,,ROOMA     40% TO ROOMA
      ADVANCE     300,60         SPEND TIME IN ROOM B
      TRANSFER    .35,,LEAVE     35% THEN LEAVE
ROOMA ADVANCE     220,40         SPEND TIME IN ROOM A
LEAVE TERMINATE                  LEAVE THE GALLERY
      GENERATE    3600*4         SIMULATE FOR 4 HOURS
      TERMINATE   1              ALL DONE
      START       1              START
      END
```

2. Change the block

```
LEAVE TERMINATE                  LEAVE THE GALLERY
```

to

```
LEAVE TERMINATE   1              LEAVE THE GALLERY
```

Delete the blocks

```
      GENERATE    3600*4         SIMULATE FOR 4 HOURS
      TERMINATE   1              ALL DONE
```

The START block is now

```
      START     200

   3.  SIMULATE
      GENERATE  35,15        PEOPLE ARRIVE
      TRANSFER  .3,,ALONG    30% GO ALONG SIDE
      TRANSFER  .25,,FASTON  25% WILL WALK/RUN
      ADVANCE   55           SIDEWALK MOVES
      TRANSFER  ,DONE        ALL DONE FOR SOME
FASTON ADVANCE  40,12        WALK/RUN ON SIDEWALK
DONE  TERMINATE              END OF SIDEWALK
ALONG ADVANCE   60,32        WALK/RUN ALONG SIDE
      TERMINATE              END OF SIDEWALK
      GENERATE  3600         SIMULATE FOR AN HOUR
      TERMINATE 1            TIMER TRANSACTION LEAVES
      START     1            START PROGRAM
      END
```

4. Change the block

```
DONE  TERMINATE              END OF SIDEWALK
```

to

```
DONE  TERMINATE  1           END OF SIDEWALK
```

Remove the two timer transaction blocks and change the START 1 block to:

```
      START 100.

   5.  SIMULATE
      GENERATE  10,3.2       CARS ENTER AT A
      ADVANCE   20,8         TRAVEL TO B
      TRANSFER  .3,,OUT      70% TURN TO C
      ADVANCE   15,6         TRAVEL TO C
      TRANSFER  .26,,OUT     74% TURN TO D
      ADVANCE   14,3.1       TRAVEL TO D
      TRANSFER  .69,,OUT     31% TURN TO E
      ADVANCE   12,3         TRAVEL TO E
      TERMINATE              CAR LEAVES
OUT   TERMINATE              CAR LEAVES
```

```
        GENERATE    3600*8          TIMER TRANSACTION
        TERMINATE   1               END OF SIMULATION
        START       1               BEGIN SIMULATION
        END
```

6.
```
        SIMULATE
        GENERATE    10,5            CUSTOMERS ARRIVE
        TRANSFER    .2,,DOWN        20% WANT TWO PAPERS
        ADVANCE     5,3             REST BUY ONE PAPER
        TERMINATE   1               PEOPLE LEAVE NEWSSTAND
DOWN    ADVANCE     7,3             BUT TWO PAPERS
        TERMINATE   2               PEOPLE BUY TWO PAPERS
        START       100             RUN FOR 100 PAPERS
        END
```

The time to run the above was 832.3134 time units.

7.
```
        SIMULATE
        GENERATE     ,,,2           CUSTOMERS WAITING
        TRANSFER     ,NEXT          SHOP OPENS - BUY PAPERS
        GENERATE     10,5           CUSTOMERS ARRIVE
NEXT    TRANSFER     .2,,DOWN       20% WANT TWO PAPERS
        ADVANCE      5,3            REST BUY ONE PAPER
        TERMINATE    1              PEOPLE LEAVE NEWSSTAND
DOWN    TERMINATE    2              PEOPLE BUY TWO PAPERS
        START        100            RUN FOR 100 PAPERS
        END
```

Now the time to sell 100 papers is reduced to 814.1544 seconds.

8.
```
        SIMULATE
        GENERATE     1
        TRANSFER     .5,,THIRD      50% GO TO SECOND FLOOR
        ADVANCE      20,10          WALK FROM FIRST TO SECOND FLOOR
        TERMINATE                   ARRIVE SECOND FLOOR
THIRD   TRANSFER     .5,,FOURTH     25% WANT TO GO TO THIRD FLOOR
        ADVANCE      20,10          WALK FROM FIRST TO SECOND FLOOR
        ADVANCE      20,10          WALK FROM SECOND TO THIRD FLOOR
        TERMINATE                   ARRIVE THIRD FLOOR
```

```
FOURTH ADVANCE    20,10        WALK FROM FIRST TO SECOND FLOOR
       ADVANCE    20,10        WALK FROM SECOND TO THIRD FLOOR
       ADVANCE    20,10        WALK FROM THIRD TO FOURTH FLOOR
       TERMINATE  1            ARRIVE FOURTH FLOOR
       START      100
       END
```

Note that the above problem assumes that people who go to the second floor go only there and the same for floors three and four. The solution gives: 506 people entered the stairs. 257 went to the second floor; 137 to the third and 112 to the fourth. The program did not end until 100 people arrived at the fourth floor.

9.

```
       SIMULATE
       GENERATE   ,,,20        20 PEOPLE TO TAKE TEST
BACK   ADVANCE    5,2          TAKE TEST
       TRANSFER   .2,,DONE     20% PASS
       ADVANCE    10           WAIT 10 MINUTES
       TRANSFER   ,BACK        TAKE TEST AGAIN
DONE   TERMINATE  1            LEAVE SYSTEM
       START      20
       END
```

The time for the people to pass the test was 280.6469. This is quite large. The number of people taking the test, including repeats was 115. Do you think that the time of 280.6469 means much? Interestingly enough, most people would not be able to guess the "solution" to this problem. The answer depends on doing the problem many times and forming confidence limits. Actually, it would be very easy to run it many times using DO loops which are a part of GPSS/H. These are covered in a later chapter.

10.

```
       SIMULATE
       GENERATE   20,10         CARS ARRIVE
       TRANSFER   .90,,NEXT1    10% HAVE ONE PERSON
       TERMINATE  1             ONE PERSON GETS OUT
NEXT1  TRANSFER   .666,,NEXT2   30% HAVE TWO PEOPLE
*****************************************************
* NOTICE THAT THE PERCENTAGE USED ABOVE IS 66.6%. THIS
* IS BECAUSE AFTER 10% OF THE CARS DROP OFF THEIR ONE
* PASSENGER, THE REMAINING NUMBERS THAT HAVE TWO PEOPLE
* WOULD BE 30/90 AND 60/90 WITH MORE (3 OR 4)
*****************************************************
```

```
      TERMINATE   2             TWO PEOPLE ARRIVE
NEXT2 TRANSFER    .25,,NEXT3    45% HAVE FOUR PEOPLE
      TERMINATE   3             THREE PEOPLE ARRIVE
NEXT3 TERMINATE   4             FOUR PEOPLE ARRIVE
      START       1000
      END
```

The time for the simulation to run was 7439.7176 time units. 380 cars came; 35 had 1 person; 119 had 2; 175 had 3 and 51 had 4.

11.
```
      SIMULATE
      GENERATE    15,5          PEOPLE ARRIVE AT COUNTER
      TRANSFER    .3,,AONLY     30% WANT A ONLY
      TRANSFER    .857,,AANDB   60% WANT A & B
      TERMINATE   1             A AND B
AANDB TERMINATE   1             B ONLY
AONLY TERMINATE   1             A ONLY
      START       100
      END
```

The simulation ran for 1484.4985 time units. The number of customers who went to A only was 27; B only 13 and A and B was 60.

12.
```
      SIMULATE
      GENERATE    20,5          NEW CONTAINERS ARRIVE
BACK  ADVANCE     40,10         MAKE DELIVERIES
      TRANSFER    .02,,DAMAGE   1 OF 50 DAMAGED
      ADVANCE     40,10         RETURN FOR RE-USE
      TRANSFER    ,BACK         READY FOR ANOTHER DELIVERY
DAMAGE TERMINATE                DAMAGED CONTAINER
      GENERATE    480           SIMULATE FOR 480 TIME UNITS
      TERMINATE   1
      START       1
      END
```

After 480 minutes, there were 24 containers that had been made. At an average of 3 per hour, this is the expected number. Two of these were damaged and discarded. Thus, there are 22 in the system.

13.
```
      SIMULATE
      GENERATE    10,5           ARRIVE AT ENTRANCE 1
```

```
      ADVANCE     15,5          MOVE ALONG CORRIDOR
      TRANSFER    ,DOWN         MERGE WITH OTHER PEOPLE
      GENERATE    5,2           ARRIVE AT ENTRANCE 2
      ADVANCE     20,8          MOVE ALONG CORRIDOR
DOWN  ADVANCE     5,3           MOVE ALONG THIRD CORRIDOR
      TRANSFER    .60,,NORTH    60% GO TO NORTH PLATFORM
      TERMINATE   1             REST GO TO SOUTH PLATFORM
NORTH TERMINATE
      START       100
END
```

At the end of the simulation, 81 people had arrived from entrance 1 and 160 from entrance 2. The simulation ran for 821.4095 time units.

```
  14. SIMULATE
      GENERATE    50,10         MANUFACTURE A PART
UPTOP ADVANCE     30,10         INSPECT A PART
      TRANSFER    .85,,PASS     85% PASS INSPECTION
      TRANSFER    .05,,SCRAP    5% OF THE REST ARE SCRAP
      ADVANCE     100,30        REWORK PARTS
      TRANSFER    ,UPTOP        BACK FOR RE-INSPECTION
PASS  TERMINATE   1             GOOD PART LEAVES
SCRAP TERMINATE                 SCRAP PILE
      START 100
      END
```

The simulation ran for 5091 time units. 13 parts needed re-working and 0 were scrapped.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 5
The ADVANCE block

The ADVANCE block

The ADVANCE Block is used to `hold up' a transaction while service is being performed. There are several forms of it. They are:

(a) `ADVANCE A`
(b) `ADVANCE A,B`

where A and B can be positive integers or variables.

In (a), the transaction will be held up a time equal to Operand A. So,

```
ADVANCE 5
ADVANCE 100
```

will hold up the transaction until 5 time units have passed in the first case and until 100 time units have passed in the second.

In (b), the transaction will be delayed by a time value from between the interval A - B and A + B. The end points are not included. However, the time returned will have a four place decimal. Each time will have equal probability of occurrence. Thus,

```
ADVANCE 12,3
```

will hold a transaction until a time between the interval 9.0001 and 14.9999 has elapsed. Each of the possible times will happen with equal probability. Later, we shall learn how to use any statistical distribution in the ADVANCE block.

When a transaction enters an ADVANCE block it is taken off the current events chain and put on a chain known as the future events chain or FEC. It will remain there until the time is reached that is given by the operand in the ADVANCE block. It is then put back onto the current events chain for further movement through the system.

We have been using ADVANCE blocks in many of our previous examples. They are one of the

easiest GPSS/H blocks to understand and use.

Example 5.1

There are 15 people in the class learning GPSS. They all start writing a program at the same time. It takes each person 12 ± 3 minutes to write the program. Only 25% of the programs will run successfully the first time. When a program has an error, it takes 8 ± 3.5 minutes to do the de-bugging. For the second (and subsequent de-buggings) programs will run successfully 35% of the time. How long does it take for the whole class to finish writing the program?

For the first time we now have had enough GPSS to understand the complete program. The program listing is given as:

```
      SIMULATE
      GENERATE    ,,,15         15 PEOPLE WRITE PROGRAM
      ADVANCE     12,3          DO WRITING
      TRANSFER    .25,,DONE     25% WORK FIRST TIME
AGAIN ADVANCE     8,3.5         DE-BUG PROGRAM
      TRANSFER    .65,,AGAIN    35% WORK ON RE-DO
DONE  TERMINATE   1             PERSON LEAVES ROOM
      START       15
      END
```

A portion of the output of the program looks as follows:

```
RELATIVE CLOCK: 63.7013 ABSOLUTE CLOCK: 63.7013

BLOCK  CURRENT  TOTAL
1               15
2               15
3               15
AGAIN           40
5               40
DONE            15
```

What happened is as follows. At time 0, 15 student transactions were created. Each, in turn, entered the ADVANCE block where each was assigned a time to be on the future events chain. For example, suppose that the students are referred to as A, B, .. , O. Suppose that the following times were from the simulation:

Student	Time
A*	12.5667
B	13.4338
C	9.5581
D	9.0809
E*	10.1210
F	12.7301
G	11.3542
H*	9.3433
I	14.8162
J	14.1931
K*	13.2222
L*	9.4538
M	9.9444
N	12.1288
O	11.6687

The asterisk next to students A, E, H, K and L refer to the fact that each was successful in running the program the first time through. At time 9.5581 student C is taken off the future events chain and enters the TRANSFER .25,,DONE block. Notice that this block was entered 15 times. Since students A, E and H were successful the first time, they were transferred to the DONE TERMINATE 1 block. The rest went to the BACK ADVANCE 8,3.5 block where they were again put on the future events chain. As, each came off and entered the TRANSFER .65,,AGAIN block. If they were successful in running the program, they were transferred to the block DONE TERMINATE 1. Eventually, all 15 students were finished. The time for all of the students to finish was found to be 63.7013 time units.

The results from this example have to be treated with care. The program was run only one time with a particular set of random numbers. What would happen if different random numbers had been used? To answer this, the program was run for 19 additional times using different random numbers. A summary of the results of these 20 simulations are:

simulation number	time to finish
1	63.70

2	71.72
3	57.27
4	103.59
5	59.12
6	52.84
7	74.61
8	62.46
9	73.75
10	80.05
11	100.33
12	78.83
13	62.40
14	45.10
15	73.94
16	64.04
17	39.09
18	66.33
19	98.61
20	45.06

The times varied from a low of 39.09 to a high of 103.59. The average time was: 68.64

A CAUTION IN WRITING PROGRAMS

There is a caution to keep in mind with this and other GPSS/H programs. In this program the transactions were all created at time t = 0. They all left at time 0 and were put on the future events chain via the ADVANCE block. The ADVANCE block always admits transactions. Suppose we had a different problem and were going to generate transactions at various times as given by

```
GENERATE 12,3
```

Suppose the times for the first four of these transactions to enter the system are 11, 23, 35 and 44. If the block after the GENERATE block will not allow a transaction to enter it, (some blocks will only allow one transaction at a time to enter them, others may not allow any transactions to enter depending on a particular condition), what happens is that the transaction remains in the

GENERATE block until the next block will allow it to enter. This may seem to be all right but, because the transaction cannot leave the GENERATE block when it was originally scheduled to, the subsequent transactions are also delayed from leaving. Suppose that the third transaction cannot enter the next block but must remain in the GENERATE block for 5 time units. This means that it cannot leave until t = 40. The effect of this on the fourth (and subsequent transactions) is to shift them all 5 time units forward before they leave. This is normally incorrect. When a transaction is scheduled to leave the GENERATE block, a block that will always accept it should be provided. One way around this is to have

```
GENERATE  12,3
ADVANCE   0
```

The ADVANCE block used here is a dummy block. It holds the transactions from the GENERATE block for zero time units. The only effect of it is to allow transactions to leave the GENERATE block at the times they were scheduled to leave. Keep this in mind for future programs.

Example 5.2

People arrive and an art exhibit every 4 ± 2 minutes. There are three rooms to view. Everyone goes to the entrance where it takes 5 ± 3 minutes to pick up a program and pay an entry fee. 80% of the people then go to room A, the rest to room B. Once a person misses a room, he or she does not go back to view it. When people leave room A 75% go to room B, the rest go to room C. Everyone who leaves room B will also go to room C. It takes 15 ± 3 minutes to view room A, 22 ± 6 minutes to view room B and 12 ± 3 minutes to view room C. When a person leaves room C, he or she leaves the exhibit. Simulate for 100 people viewing the exhibit.

Solution

The program for the problem is:

```
      SIMULATE
      GENERATE    4,2          PEOPLE ARRIVE AT THE EXHIBIT
      ADVANCE     5,3          SPEND TIME IN THE ENTRANCE
      TRANSFER    .2,,ROOMB    80% TO ROOM A
      ADVANCE     15,3         VIEW ROOM A
      TRANSFER    .25,,ROOMC   75% TO ROOM B
ROOMB ADVANCE     22,6         VIEW ROOM B
ROOMC ADVANCE     12,3         VIEW ROOM C
      TERMINATE   1            LEAVE THE EXHIBIT
```

```
START      100
END
```

Exercise 5.1

1. Consider Example 2 again. Determine how long it will take until 100 people have viewed all three rooms.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 6
QUEUE/DEPART blocks

The QUEUE Block

Since GPSS/H is used so often for simulation of systems where queues are formed at many places, it is natural to learn how the language handles queues. A discussion of queues and the mathematical theory associated with them can be found in any textbook on Operations Research. In fact, there are complete books devoted to this important topic. One thing becomes quite clear when one studies queuing theory: the number of queuing problems that have exact mathematical solutions is surprisingly small. This is especially so when one is dealing with a finite number of transactions, such as the case of where transactions cycle though the system.

There are many cases when a transaction will be denied access to a block during a simulation. When a transaction is to use a facility that is already in use, it is denied entry and has to remain in the block where it is presently resides. In the system being simulated, this gives rise to a queue forming. Such queues are commonly found in real life situations. These might be found in a barber shop with only one barber, a checkout counter in a grocery store, a bank with many tellers, an airport with only a few runways, etc. Often the purpose of the simulation study is to see where these queues form and how they might be eliminated or, perhaps, kept to a reasonable level. These queuing situations are handled in GPSS/H by the QUEUE block. The normal form of the QUEUE block is quite simple. It is

```
QUEUE A
```

where the Operand A is either a name (at least 3 letters and not more than 5 characters in fixed format and 8 characters in free format) or a number. It could also be a variable, as we shall learn. Thus,

```
QUEUE 1
QUEUE FIRST
QUEUE 7
QUEUE ONE
QUEUE DUMP1
QUEUE STOPHERE
```

are examples of valid QUEUE blocks but

```
QUEUE -1
QUEUE LASTONEIN
```

are not.

Sometimes you will decide to use a number rather than a name for the QUEUE block's operand. If you do choose this, the number cannot be arbitrary but will depend on the actual number of QUEUE blocks allowed in your system. Normally, at least 50 QUEUE blocks are allowed in most GPSS processors. Thus, if this is the maximum number allowed in your system, it would be all right to have

```
QUEUE 27
QUEUE 50
```

but not

```
QUEUE 123
```

Should you decide to use numbers in the operands, simply remember to start numbering the QUEUE blocks with small numbers and you should not have any problem.

Whenever a QUEUE block is used, there will automatically be certain statistics printed out when the program is finished. These were observed when previous programs were run. Suppose the QUEUE block was specified by the Operand WAIT. The output from the program might look as follows:

```
QUEUE    MAXIMUM    AVERAGE    TOTAL      ZERO
         CONTENTS   CONTENTS   ENTRIES    ENTRIES
WAIT        3         0.312      264         90

         PERCENT    AVERAGE    $AVERAGE   QTABLE    CURRENT
         ZEROS      TIME/UNIT  TIME/UNIT             CONTENTS
          34.1       5.665      8.596                   0
```

The above is actually given in the output as running across the screen. It is necessary to scroll to see it all. What each entry means is as follows:

QUEUE
WAIT — this is the name of the queue as specified by the A operand

MAXIMUM CONTENTS 3	the maximum contents of the queue at any time during the simulation was
AVERAGE CONTENTS 0.312	at any time during the simulation the average contents in the queue was 0.312
TOTAL ENTRIES 264	the number of transactions that entered the block was 264
ZERO ENTRIES 90	of the 264 transactions that entered the QUEUE block 90 of them immediately left and entered the next block
PERCENT ZEROS 34.1	the quotient 90/264
AVERAGE TIME/UNIT 5.665	for all the transactions that entered the QUEUE block, this is the average time in the block
$AVERAGE TIME/UNIT 8.596	this is the average time in the QUEUE block for only the transactions that were actually delayed and held in it
QTABLE	later, we shall see how to construct histograms of various parameters associated with the simulation. One of these is called a QTABLE. If one had been used in the simulation, its name would be here.
CURRENT CONTENTS 0	the contents of the QUEUE block at the end of the simulation

You may not want all the output, but GPSS gives it to you regardless. In a later chapter you will learn how to customize your output if so desired.

The above items are all attributes associated with having a QUEUE block. In fact, they are called Standard Numerical Attributes (SNA's). These all have reserved names. These are as follows:

SNA	meaning
Q(name) or Qn	current queue content
QA(name) or QAn	average queue contents
QC(name) or QCn	queue entry count
QM(name) or QMn	maximum queue content
QT(name) or QTn	average time spent in the queue of all entries
QX(name) or QXn	average time spent in the queue excluding the zero entries
QZ(name) or QZn	zero entries

The above SNA's can be used in the program as operands. For example, one could have:

```
ADVANCE QM(WAIT)
```

The transaction entering the ADVANCE block would be put on the FEC for a time given by the maximum queue length at the QUEUE WAIT. In Chapter 10 we will learn more uses for SNA's.

The most common reason for using a QUEUE block is to gather these statistics if the transaction is delayed waiting to use a facility.

The QUEUE block never denies entry to a transaction and so it can, in theory, contain any number of transactions. It should also be kept in mind that it is not always necessary to have a QUEUE block just because a queuing situation is to take place. As indicated, it is possible to have a second operand with the QUEUE block such as

```
QUEUE FIRST,2
QUEUE WAIT,3
```

This second operand must be a positive number. If it is used, it will affect the statistics of the QUEUE block as the B operand will cause the `total entry count' to be increased by this amount (not 1) and the 'current content' is increased by this amount.

Should you ever decide to use such a QUEUE block, you must be very careful to interpret your results accordingly. Actually, such use of the QUEUE block is very rare. In fact, there will not be any in the rest of this book.

There is another point worth mentioning concerning the QUEUE block and that is the fact that a transaction can be in more than one QUEUE block at the same time. This may seem strange but such a situation occurs in real life. Consider a major shopping center where a person has to take a number to purchase meat. The same person can elect to also take a number to purchase vegetables

while waiting for the first number to be called. Thus, the person is in two queues at the same time. There will be occasions when, for the purpose of gathering statistics, we will use the fact that a transaction can be in more than one QUEUE block at the same time. In fact, a transaction can be in even more than two QUEUE blocks at the same time. The number of QUEUE blocks a transaction can be in at the same time is dependent on the particular processor but is around 5.

The DEPART Block

If a transaction is in a QUEUE block, it must eventually leave this block. This is done by the DEPART block. It is used as the twin to the QUEUE block and it has the same operand. Thus, referring to the examples of the QUEUE block, the following would be the corresponding DEPART blocks.

```
DEPART 1
DEPART FIRST
DEPART 7
DEPART ONE
DEPART DUMP1
DEPART STOPHERE
```

The DEPART block will not be immediately after the QUEUE block but must appear in the program. (If it were immediately after QUEUE block, the QUEUE block would give meaningless statistics as the transactions would immediately enter and leave both blocks). It usually appears after one or two other blocks. These other blocks are the ones that for one reason or other, cause a queue to form. Just as with the QUEUE block it is possible to have a second operand,

```
DEPART NAME,2
```

In this case the current content of the QUEUE NAME is decreased by 2. If the transaction was a zero entry to the QUEUE, the zero entry counter is incremented by 2. As with the second operand for the QUEUE block, use of this operand is very rare.

Example 6.1

Customers arrive in Joe's barber shop at a uniform rate of one every 18 ± 5 minutes. Joe can cut hair at the rate of 16 ± 4 minutes. Simulate for 8 hours.

Solution

The solution to this involves two blocks that we haven't had yet. Both of these will be given in

the next Chapter.

```
SIMULATE
GENERATE 18,5 CUSTOMERS ARRIVE
QUEUE WAIT JOIN THE QUEUE
SEIZE BARBER USE THE BARBER
DEPART WAIT LEAVE THE QUEUE
ADVANCE 16,4 GET HAIRCUT
RELEASE BARBER FREE THE BARBER
TERMINATE LEAVE THE SHOP
GENERATE 480 TIMER TRANSACTION
TERMINATE 1 STOP SIMULATION
START 1 SIMULATION BEGINS
END
```

Selected portions of the output are given next:

```
RELATIVE CLOCK: 480.0000 ABSOLUTE CLOCK: 480.000

BLOCK CURRENT TOTAL
1             26
2             26
3             26
4             26
5     1       26
6             25
7             25
8             1
9             1

--AVG-UTIL-DURING--

FACILITY    TOTAL    ENTRIES    AVERAGE
            TIME               TIME/XACT
BARBER      0.858      26        15.847

QUEUE       MAXIMUM    AVERAGE    TOTAL    ZERO    AVERAGE    $AVERAGE
            CONTENTS   CONTENTS   ENTRIES  ENTRIES TIME/UNIT
TIME/UNIT
WAIT          1         0.040      26       17      0.746      2.155
```

During the 8 hours of the simulation, 26 customers entered the store. 17 of them did not have to wait for the barber to be free. the average content of the queue was only .04 and the maximum content of the queue was only 1. The barber was busy 85.8% of the time.

Exercises

1. Refer to Example 6.1. Suppose that the first customer arrives at exactly when the shop opens. Also, let the time to cut hair be 16 ± 7.5 minutes and the arrival rate now 18 ± 9 minutes. Do the statistics change much?

2. Two types of customers arrive at Joe's barber shop. The first type want only a haircut. They come every 35 ± 10 minutes. The second type want both a haircut and a shave. They arrive every 60 ± 20 minutes. It takes Joe 18 ± 6 minutes to give a haircut and 10 ± 2 minutes for a shave. Construct a model of the shop. Run it for 20 days of 8 hours straight. Determine if Joe is working too hard. (This is defined by the union as working more than 85% of the time).

b) Suppose Joe decides to give preference to customers who want only a haircut. How does this change the situation?

c) add other queues to the problem to gather statistics about the various queues that form in the barber shop.

3. Three types of mechanics arrive at a tool crib to check out tools. Only one clerk works at the crib. The arrival times and service times are:

type	dist of arrival time	dist of service time
1	30 ± 10	12 ± 5
2	20 ± 8	6 ± 3
3	15 ± 5	3 ± 1

Model the above for 10 days. (the times above are minutes.)

4. In an efficient shop (where no infinite queues are forming) you can show that it is more efficient for the workers to perform tasks that can be finished in the shortest time (assuming that the cost/benefit of the jobs is the same). The following example illustrates this. A tool crib receives two type of requests for service. These are called type 1 request and type 2 requests. The interarrival times and mean service times for each are:

service	arrival times (sec)	service times
type 1	520 ± 360	340 ± 90
type 2	250 ± 100	70 ± 40

Write the GPSS program with no priority for service and then with priority given to type 2 service. Suppose that the people requesting service earn $12.00/hr. Determine the savings each day by having a priority system. Also, run the program with type 1 service having priority.

Solutions

2.
```
GENERATE    35,10     HAIRCUT CUSTOMERS ARRIVE
QUEUE       JOEQ      SIT IN THE CHAIRS
SEIZE       JOE       SEIZE POOR JOE
DEPART      JOEQ      LEAVE THE CHAIR
ADVANCE     18,6      GET THE HAIRCUT
RELEASE     JOE       FREE JOE
TERMINATE             LEAVE
GENERATE    60,20     OTHER CUSTOMERS ARRIVE
QUEUE       JOEQ      SIT IN THE CHAIRS
SEIZE       JOE       SEIZE GOOD OLD JOE
DEPART      JOEQ      LEAVE THE CHAIR
ADVANCE     10,2      GET SHAVE
ADVANCE     18,6      GET HAIRCUT
RELEASE     JOE       FREE JOE
TERMINATE   LEAVE     SHOP
GENERATE    480*20    TIMER TRANSACTION
TERMINATE   1
START       1
END
```

 **Return on CONTENS**

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 7.
The SEIZE and RELEASE blocks

The SEIZE Block

In GPSS a single server is called a `facility'. This might represent the barber giving a haircut, a bank clerk who waits on customers, a checkout worker in a grocery store, etc. In order for a transaction to use a facility, a SEIZE block is used. This is quite simple to use as one form is:

```
SEIZE A
```

where the A operand is generally either a number or name, but can be a variable. Thus,

```
SEIZE 1
SEIZE ONE
SEIZE DUMP
SEIZE BARBER
SEIZE 33
```

are examples of the SEIZE block. Should a number be used for the operand of the SEIZE block, the number used must be less than the number of SEIZE blocks allowed by your processor. If you always remember numbering the blocks starting with small numbers, this should not be a problem.

When a transaction enters a SEIZE block, no other transaction can enter it until the transaction in the SEIZE block leaves. Transactions attempting to enter a SEIZE block that is already being used by another must (normally) remain in the block they are in.

Whenever a facility is used by the SEIZE block, certain statistics are automatically printed out. We have seen this in many of the programs we have already written. Since a facility can be used by only one transaction at a time, if a second transaction wishes to use it, it must wait in the previous block until the facility is free. For example, consider the blocks:

```
GENERATE  10
QUEUE     WAIT
SEIZE     JOE
DEPART    WAIT
ADVANCE   25
```

Here a transaction is generated at t = 10. It is moved to the QUEUE block and immediately attempts to use the facility JOE. Since JOE is not being used, it does so and then enters the ADVANCE block where it is put in the future events chain until a time of t = 35 (10 + 25). A second transaction is generated at time 20. It enters the QUEUE WAIT block and attempts to enter the SEIZE JOE block. Since it cannot, it is held up in the QUEUE block until JOE is free. Now suppose that you did not have the QUEUE WAIT block. Instead, suppose you had:

```
GENERATE  10
SEIZE     JOE
ADVANCE   25
```

The second transaction would have been held up in the GENERATE block until t = 35. The effect of this in GPSS would be to offset the time for when the third transaction leaves the GENERATE block from t = 30 to t = 50. (Draw a time diagram to convince yourself of this.) Since this is not what we want, this should be avoided by either a QUEUE block or a dummy ADVANCE block. A dummy ADVANCE block would look as follows:

```
      GENERATE  10
      ADVANCE
************************************
* NOTE: YOU COULD ALSO HAVE *
* ADVANCE 0 *
************************************
      SEIZE     JOE
      ADVANCE   25
```

Whenever a SEIZE block is used, certain statistics are automatically printed out at the end of the simulation. To illustrate this, recall the program of the barber in Chapter 3. The barber could cut hair in 12 ± 4 minutes and customers arrived at the rate of 15 ± 6 minutes. The program is repeated here:

```
      SIMULATE
      GENERATE   15,6      PEOPLE ENTER SHOP
      QUEUE      SEAT      TAKE A SEAT
      SEIZE      BARBER    IF BARBER FREE, BEGIN HAIRCUT
      DEPART     SEAT      LEAVE THE SEAT
      ADVANCE    12,4      RECEIVE HAIRCUT
      RELEASE    BARBER    HAIRCUT OVER, BARBER IS FREE
      TERMINATE  1         LEAVE THE SHOP
      START      200       SIMULATE FOR 200 CUSTOMERS
```

```
      END
```

The output associated with the SEIZE block is as follows:

```
            --AVG-UTIL-DURING--

FACILITY     TOTAL      AVAIL      UNAVL      ENTRIES    AVERAGE
             TIME       TIME       TIME                  TIME/XACT
BARBER       0.804                                200      12.088

             CURRENT    PERCENT    SEIZING    PREEMPTING
             STATUS     AVAIL      XACT       XACT
             AVAIL
```

The output is interpreted as follows:

FACILITY	The name of the facility was BARBER
TOTAL TIME	The facility was busy 80.4% of the time the simulation ran.
AVAIL TIME	Later we shall learn that it is possible to "shut down" a facility and make it unavailable. This would show the time the facility was not available.
UNAVL TIME	The time the facility was not available.
ENTRIES	There were 200 entries. Recall that the program ran until the barber finished 200 haircuts.
AVERAGE TIME/XACT	The average time the SEIZE block was used by a transaction was 12.088. The time to give a haircut was 12 ± 4 minutes. The mean of this distribution is 12.
CURRENT STATUS AVAIL	The facility is currently available.
PERCENT AVAIL	Since the facility was never made unavailable there is not entry here.

SEIZING XACT	If the facility was in use at the end of the program, the transaction number seizing it would be given here.

Just as with the QUEUE block, the above are SNA's of the system and each will be referred to by a special reserved name. These will be covered in Chapter 10.

SNA's Associated with the SEIZE Block

The SNA's associated with the SEIZE Block are given below:

SNA	meaning
F(name) or Fn	this will be 1 if the facility is currently being used; else it's value is 0
FC(name) or FCn	number of time the facility has been captured
FR(name) or FRn	utilization of the facility in parts per thousand
FT(name) or FTn	average holding time

Note that the output from the program given the utilization as a decimal but the SNA FR is in parts per thousand. Thus, if one had used the block:

```
ADVANCE FR(MACH1)
```

and the utilization of the facility MACH1 was .432 when the transaction entered the ADVANCE block, the transaction would be placed on the FEC for 432 time units.

The RELEASE Block

When a facility is used via the SEIZE block, it must be eventually freed for other transactions to use it. This is done by means of the `twin' of the SEIZE block, the RELEASE block. Some forms are:

```
RELEASE JOE
RELEASE 1
RELEASE BARBER
RELEASE CAR1
```

At this point you may wish to go back to some of the previous exercises and examine the output whenever a facility is used in the program. It is very common in our programs to have a sequence of blocks such as:

```
QUEUE     HERE
SEIZE     THERE
DEPART    HERE
ADVANCE   20,3
RELEASE   THERE
```

This sequence should be carefully examined and understood as it is repeated a great many time in GPSS programs.

We are now in a position to understand nearly all of the programs we have written so far in the previous Chapters. The next few examples will enable us to use what we have learned so far. Each should be carefully studied and understood.

Example 7.1

Cars come to a garage for minor repairs. These are of two types, routine and non-routine. There is only one mechanic who does both types of repairs. 70% of the cars that enter need only routine repairs while the rest are in for non-routine repairs. Cars arrive every 28 ± 7 minutes. Non-routine repairs take 45 ± 15 minutes while routine repairs take only 18 ± 6 minutes. The single mechanic is claiming that he is overworked. His union defines this as working more than 85% of the time. Is he justified in his claim? Simulate for 5 days of 480 minutes each. Ignore the fact that the worker leaves at the end of each shift and the effect of weekends.

Solution

The program to do this is:

```
SIMULATE
GENERATE    28,7          CARS ARRIVE FOR REPAIRS
TRANSFER    .70,,MINOR    70% NEED MINOR REPAIRS
QUEUE       WAIT          JOIN QUEUE
SEIZE       MECH          IS MECHANIC FREE?
DEPART      WAIT          YES, LEAVE QUEUE
ADVANCE     45,15         MAJOR REPAIR TAKES PLACE
RELEASE     MECH          FREE THE MECHANIC
TERMINATE                 LEAVE GARAGE
```

```
MINOR QUEUE      WAIT        JOIN QUEUE
      SEIZE      MECH        IS MECHANIC FREE?
      DEPART     WAIT        YES, LEAVE QUEUE
      ADVANCE    18,6        MINOR REPAIR TAKES PLACE
      RELEASE    MECH        FREE MECHANIC
      TERMINATE              LEAVE GARAGE
      GENERATE   480*5       TIMER TRANSACTION ARRIVES
      TERMINATE  1
      START      1
END
```

Selected portions of the output are:

```
Simulation begins.

RELATIVE CLOCK: 2400.0000 ABSOLUTE CLOCK: 2400.0000

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               84     11              56
2               84     12              56
3               28     13              56
4               28     14              56
5               28     15              1
6      1        28     16              1
7               27
8               27
MINOR           56
10              56

        --AVG-UTIL-DURING--

FACILITY    TOTAL      ENTRIES      AVERAGE
            TIME                    TIME/XACT
MECH        0.940        84          26.867

QUEUE     MAXIMUM    AVERAGE    TOTAL     ZERO
          CONTENTS   CONTENTS   ENTRIES   ENTRIES
WAIT         4         1.393      84        14
```

The results are interpreted as follows. During the 5 days of simulated time 84 cars came in for repairs. The mechanic was busy 94% of the time. This is beyond the 85% called for so he has a legitimate complaint of being overworked.

Example 7.2

The owner of a small gold mine is wondering if he has the right number of trucks to haul the ore. Figure 7.1 gives a sketch of the operation. Trucks are loaded by a single shovel and then travel to the a processing plant where they dump and cycle back to the shovel. Only 1 truck at a time can be loaded and at the processing plant there is no such limitation since the trucks dump the ore into a pile. It takes 3.5 ± 1.25 minutes to load a truck, 6 ± 2.75 minutes to haul to the dump, 2.1 ± .4 minutes to dump the ore and 4.8 ± 1.8 minutes to return. Financial data associated with this operation is as follows:

Item	
Truck driver's salary	$15.75/hour
Cost of running the shovel, dump, etc.	$275 per 8 hr day
Profit per load (after all other expenses)	$25.50

Determine the correct number of trucks to have in the mine.

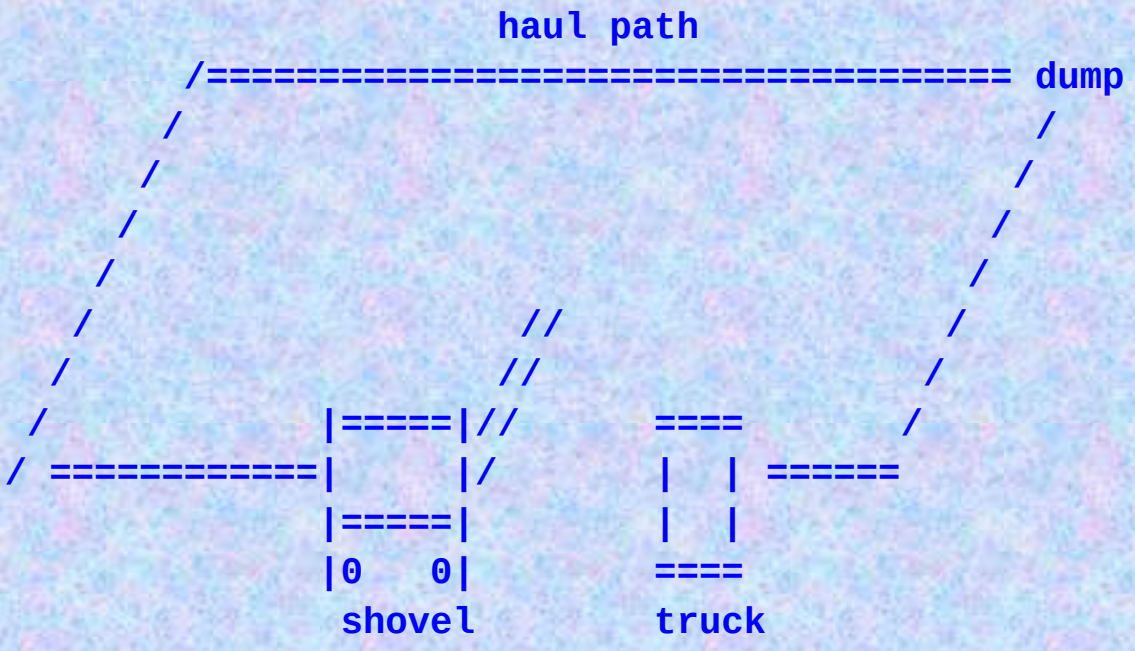


Figure 7.1 Sketch of Mine and Trucks

The computer program that was used to obtain a solution for this problem is:

```
      SIMULATE
      GENERATE    ,,,2          PUT TRUCKS IN THE MINE
UPTOP QUEUE       WAIT          QUEUE AT THE SHOVEL
      SEIZE       SHOVEL        USE THE SHOVEL
      DEPART      WAIT          LEAVE THE QUEUE
      ADVANCE     3.5,1.25      LOAD A TRUCK
      RELEASE     SHOVEL        FREE THE SHOVEL
      ADVANCE     6.5,2.75      TRAVEL TO DUMP
      ADVANCE     2.1,.4        DUMP A LOAD OF ORE
      ADVANCE     4.8,1.8       RETURN TO SHOVEL
      TRANSFER    ,UPTOP        JOIN QUEUE AGAIN
      GENERATE    480           SIMULATE FOR A SINGLE SHIFT
      TERMINATE   1
      START       1
      END
```

The program was run repeatedly for 2 trucks, then 3 trucks, etc. up to 7 trucks. The results of the simulations gave the following:

Table 7.1. Results of Multiple Simulations

no. trucks	loads/day	% util shovel	avg. queue
2	56	.409	.031
3	82	.602	.104
4	107	.778	.241
5	127	.926	.535
6	136	.995	1.200
7	136	1.000	2.194

All that is needed from the simulation results is the loads/day but it is instructive to examine the other data. With only 2 trucks the shovel is busy only 40.9% of the time. The average queue length is only .031. As the number of trucks in the mine is increased the utilization of the shovel increases until it is working 100% of the time. The addition of a 7th truck has no effect on the mine other than to add another truck to the queue. In order to solve this problem, the cost data needs to be used. Table 7.2 gives the results of these calculations.

Table 7.2. Results of Calculations

no. trucks	loads/day	cost of drives	fixed costs	profit
2	56	$252	$275	$339
3	82	$378	$275	$620
4	107	$504	$275	$875
5	127	$630	$275	$1,058
6	136	$756	$275	$1,083
7	136	$882	$275	$954

The optimum number of trucks to have is 6 for a maximum daily profit of $1,083. Notice, however, that there would not be much of a difference in profit if 5 trucks were used.

Exercises

1. A factory which formerly produced only widgets is branching out into the production of squidgets. To make each squidget, a person needs to assemble various parts. These take 30 ± 8 minutes to assemble each squidget. Then the squidgets need to be "fired". There is only one firing machine and so only one squidget can be fired at a time. Firing takes 8 ± 3 minutes per squidget. Each squidget produced earns a tidy profit of $6.00. The firing machine costs you $40/day no matter what (fixed costs). You pay your workers $5/hr. How many workers do you hire?

2. In the previous problem the assemblers initially began assembling a squidget. Suppose the following were the initial conditions (four assemblers total) :

three assemblers are just beginning to assemble
one is just beginning to use the oven

Change the program written for Exercise 1. to include the above.

3. Change Exercise 1 to reflect the following initial conditions for 4 assemblers:

one assembler is beginning to assemble
one has 10 minutes to go before completing an assemble
one has 3 minutes to go before finishing with the oven
one is waiting to use the oven

4. (from Schriber, 1974). Two types of customers arrive at Joe's barber shop. The first type want only a haircut. They come every 35 ± 10 minutes. The second type want both a haircut and a shave. They arrive every 60 ± 20 minutes. It takes Joe 18 ± 6 minutes to give a haircut and 10 ± 2 minutes for a shave. Construct a model of the shop. Run it for 20 days of 8 hours straight. Determine if Joe is working too hard. (This is defined by the union as working more than 85% of the time).

b) Suppose Joe decides to give preference to customers who want only a haircut. How does this change the situation?

c) add other queues to the problem to gather statistics about the various queues that form in the barber shop.

5. Three types of mechanics arrive at a tool crib to check out tools. Only one clerk works at the crib. The arrival times and service times are:

type	dist of arrival time	dist of service time
1	30 ± 10	12 ± 5
2	20 ± 8	6 ± 3
3	15 ± 5	3 ± 1

a) Model the above for 20 straight shifts. (the times above are minutes.) Each shift lasts for 480 minutes.

b) Suppose that preference is given to mechanics who take the least time for service, in this case type 3 mechanics. Change the program to determine is the solution changes.

Solutions

1. The GPSS program that can be used to solve the problem is given below. This is for 3 workers. You will have to run the program for 4, 5, 6, etc workers. Doing the computations after each run yields the following:

workers	profit
3	60
4	79
5	93

6	80
7	44

```
        SIMULATE
        GENERATE    ,,,3        PROVIDE 3 WORKERS
UPTOP   ADVANCE     30,8        IT TAKES 30 ± 8 MIN TO WORK
        SEIZE       FIRE        USE THE FURNACE
        ADVANCE     8,3         FIRE A SQUIDGET
        RELEASE     FIRE        FREE THE FURNACE
        TRANSFER    ,UPTOP      GO BACK TO DO ANOTHER JOB
        GENERATE    4800        SIMULATE FOR 10 SHIFTS
        TERMINATE   1           STOP SIMULATION
        START       1           START THE SIMULATION
        END                     END OF COMPILING
```

2.
```
        GENERATE    ,,,1        PROVIDE ONE ASSEMBLER
        TRANSFER    ,OVEN       SEND HIM TO THE OVEN
        GENERATE    ,,,3        PROVIDE 3 ASSEMBLERS
BACK    ADVANCE     30,5        ASSEMBLE A SQUIDGET
OVEN    SEIZE       OVEN        USE THE OVEN
        ADVANCE     8,2         FIRE A SQUIDGET
        RELEASE     OVEN        FREE THE OVEN
        TRANSFER    ,BACK
```

3.
```
        SIMULATE
        GENERATE    ,,,1        ONE TO BEGIN ASSEMBLING
        TRANSFER    ,UPTOP      SEND TO ASSEMBLE AREA
        GENERATE    ,,,1        ONE IS ASSEMBLING
        ADVANCE     10          THIS WILL TAKE 10 MINUTES
        TRANSFER    ,DOWN1      SEND TO OVEN
        GENERATE    ,,,1        ONE IS USING OVEN
        SEIZE       FIRE        USE THE OVEN
        ADVANCE     3           WAIT FOR THREE MINUTES
        RELEASE     FIRE        FREE THE OVEN
        TRANSFER    ,UPTOP      BACK TO DO ANOTHER
        GENERATE    ,,,1        ONE TO WAIT FOR OVEN
        TRANSFER    ,DOWN1      GO TO THE OVEN
UPTOP   ADVANCE     30,8        IT TAKES 30 ± 8 MINUTES TO WORK
```

```
DOWN1 SEIZE      FIRE        USE THE FURNACE
      ADVANCE    8,3         FIRE A SQUIDGET
      RELEASE    FIRE        FREE THE FURNACE
      TRANSFER   ,UPTOP      GO BACK TO DO ANOTHER JOB
      GENERATE   4800        SIMULATE FOR 10 SHIFTS
      TERMINATE  1           STOP SIMULATION
      START      1           START SIMULATION
      END                    END OF COMPILING
```

4.

```
      GENERATE   35,10       HAIRCUT CUSTOMERS ARRIVE
      QUEUE      JOEQ        SIT IN THE CHAIRS
      SEIZE      JOE         SEIZE POOR JOE
      DEPART     JOEQ        LEAVE THE CHAIR
      ADVANCE    18,6        GET THE HAIRCUT
      RELEASE    JOE         FREE JOE
      TERMINATE  LEAVE
      GENERATE   60,20       OTHER CUSTOMERS ARRIVE
      QUEUE      JOEQ        SIT IN THE CHAIRS
      SEIZE      JOE         SEIZE GOOD OLD JOE
      DEPART     JOEQ        LEAVE THE CHAIR
      ADVANCE    10,2        GET SHAVE
      ADVANCE    18,6        GET HAIRCUT
      RELEASE    JOE         FREE JOE
      TERMINATE              LEAVE SHOP
      GENERATE   9600        TIMER TRANSACTION
      TERMINATE  1
      START      1
      END
```

5.

a)

```
      SIMULATE
      GENERATE   30,10       TYPE 1 ARRIVES
      QUEUE      WAIT        WAIT IN QUEUE
      SEIZE      CLERK       USE THE CLERK
      DEPART     WAIT        LEAVE THE QUEUE
      ADVANCE    12,5        CLERK GETS TOOL
      RELEASE    CLERK       FREE THE CLERK
      TERMINATE  LEAVE       THE CRIB AREA
      GENERATE   20,8        TYPE 2 ARRIVES
```

```
QUEUE       WAIT        WAIT IN QUEUE
SEIZE       CLERK       USE THE CLERK
DEPART      WAIT        LEAVE THE QUEUE
ADVANCE     6,3         CLERK GETS TOOK
RELEASE     CLERK       FREE THE CLERK
TERMINATE   LEAVE       THE CRIB AREA
GENERATE    15,5        TYPE 3 ARRIVES
QUEUE       WAIT        WAIT IN QUEUE
SEIZE       CLERK       USE THE CLERK
DEPART      WAIT        LEAVE THE QUEUE
ADVANCE     3,1         CLERK GETS TOOK
RELEASE     CLERK       FREE THE CLERK
TERMINATE   LEAVE       THE CRIB AREA
GENERATE    480         ONE SHIFT PASSES
TERMINATE   1
START       20          20 DAYS
END
```

b). The block GENERATE 15,5 is changed to

```
GENERATE 15,5,,1
```

No other changes are needed.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 8
The ENTER and LEAVE blocks

Multiple Servers - the ENTER Block

It often happens that the system being studied has multiple servers which are identical. Transactions that attempt to use these will be denied access if the servers are all busy. They do not wait at each server but are held on the CEC in the previous block until one of the servers is free. Figure 8.1 illustrates this situation.

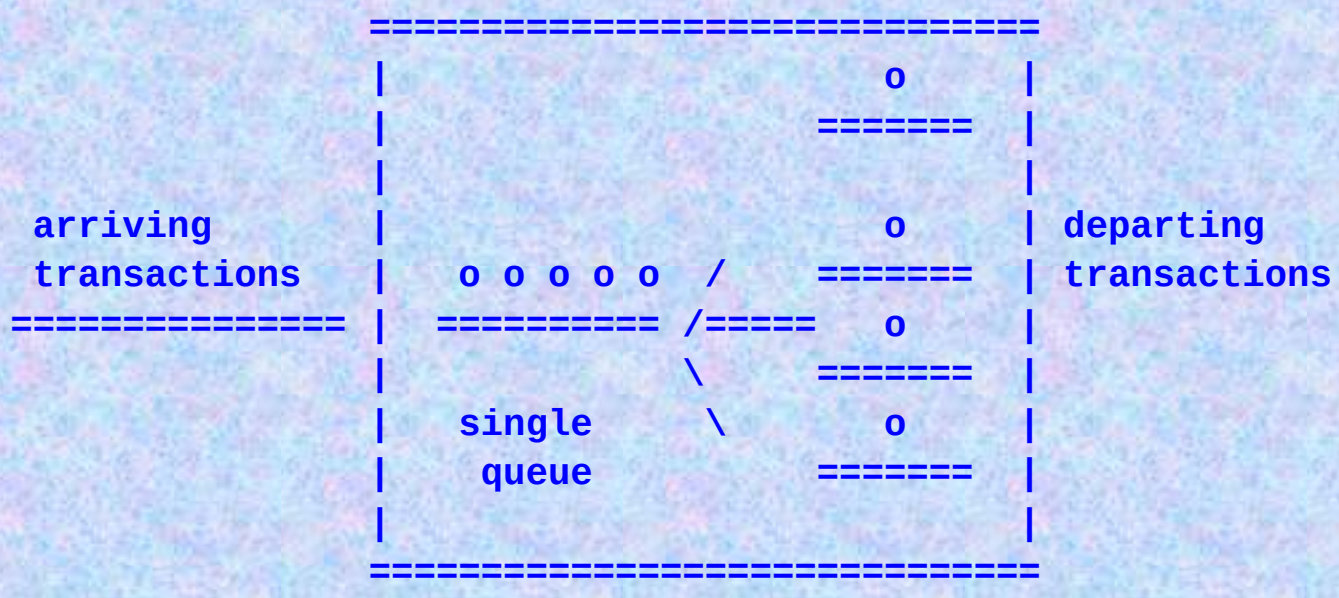


Figure 8.1 Sketch of multiple servers.

Examples of these might be the multiple berths for loading or unloading facilities in a port for ships, two barbers in a shop, six tellers in a bank, etc. GPSS handles these by means of a combination of a block and a statement. The block is used in the program for the transaction to actually use one of the parallel servers. The statement tells the processor how many of these servers are available. These servers will be referred to as `storages'. The block for the transaction in the programs is the ENTER block. One form of the ENTER block is:

```
ENTER A
```

where A is the operand. This can be a name or a number. If it is a number, it cannot exceed the

maximum number of such ENTER blocks allowed in your system. It is also possible to have this operand be a variable as we shall see later.

Examples of the ENTER block are:

```
ENTER DUMP
ENTER 2
ENTER TWO
```

The operands DUMP, 2, and TWO refer to the names of the multiple serves, or storages, as we shall now refer to them. There is no confusion in using the storages named Dump and TWO but care should be exercised with the one named "2". This does not refer to the number of storages associated with the block ENTER 2. This number is specified by a separate statement. It is quite possible that the number of storages associated with each of the above blocks might be 4 for DUMP, 3 for "2" and 4 for THREE. We will shortly learn how to specify the number of these. It is possible to have a second or B operand with the ENTER block such as

```
ENTER HARBOR,2
```

In this case two storages are used. This might represent a large ship entering a harbor and two tug boats (in this case, the tug boats are the storages) are needed to transport it in. Most of the time only one of the storages is taken each time a transaction uses the ENTER block. However, there will be times when the second operand will come in very handy.

Defining the Number of Multiple Servers - the STORAGE Statement

Normally, when there are to be multiple servers, the number of these must be specified. This is done by the STORAGE statement. There are two forms of it. The form may look a bit strange but if the programmer remembers that it is used in connection with the ENTER block, it is easy to write. Suppose there are to be 3 barbers in a shop. The customer will select one of them via the ENTER block.

```
ENTER BARBER
```

How the computer knows that there are 3 barbers is specified by the STORAGE statement. This must be placed before the ENTER block. Normally, for convenience, it is placed near the top of the program before any GENERATE block. It can have several forms. These are:

```
(a) STORAGE S(BARBER),3
```

or

```
(b) BARBER STORAGE 3
```

Both (a) and (b) are identical. When the block

```
ENTER BARBER
```

is encountered, the transaction will be able to enter it if there are less than 3 transactions in it already. If the transaction cannot move forward, it is held on the CEC time until a later scan when it can be moved. The STORAGE in (b) is identical. The reason (a) is used more often than (b) is because it is possible to have more than one STORAGE defined using (a). This is done as follows:

```
STORAGE S(BARBER),3/S(JOE),12/S(BILLY),7
```

The storage of BARBER is 3; that of JOE is 12 and BILLY is 7. If numbers are used for storages the form of the STORAGE statement is somewhat simplified:

```
STORAGE S1,2/S4,7
```

Since an operand can also be a number, the second STORAGE statement defines the storage of 1 to be 2 and of 4 to be 7. Whenever you use the ENTER block for parallel entities, you will obtain certain output when the program is over. Consider the following example.

Example 8.1

A barber shop has two identical workers. They can cut hair at a rate of 13 ± 5.5 minutes. Customers arrive every 7 ± 2.6 minutes. Simulate for an 8 hour's day. The program to do this is:

```
SIMULATE
STORAGE    S(BARBER),2    PROVIDE TWO BARBERS
GENERATE   7,2.6          CUSTOMERS ARRIVE
QUEUE      WAIT           WAIT FOR BARBER
ENTER      BARBER         USE A BARBER
DEPART     WAIT           LEAVE SEAT
ADVANCE    13,5.5         GET HAIRCUT
LEAVE      BARBER         FREE A BARBER
TERMINATE                 LEAVE SHOP
GENERATE   480            TIMER TRANSACTION
```

```
      TERMINATE  1
      START      1
      END
```

The output from the program will include the following:

```
RELATIVE CLOCK: 480.0000 ABSOLUTE CLOCK: 480.0000

BLOCK  CURRENT  TOTAL
1               68
2               68
3               68
4               68
5      2        68
6               66
7               66
8               1
9               1

          --AVG-UTIL-DURING--

STORAGE   TOTAL   AVAIL    UNAVL   ENTRIES   AVERAGE   CURRENT
          TIME    TIME     TIME               TIME/UNIT STATUS
BARBER    0.966                       68       13.635    AVAIL

          PERCENT   CAPACITY   AVERAGE   CURRENT   MAXIMUM
          AVAIL                CONTENTS  CONTENTS  CONTENTS
          100.0        2        1.932       2         2
```

The interpretation of the above is as follows:

TOTAL TIME	The storage BARBER was busy 96.6% of the time. This is for both the barbers. It is not possible to tell the percentage of time each was busy.
AVAIL TIME	It is possible to "shut down" the storages using a block to be introduced later. If a storage is shut down, the time available is given here.
UNAVL TIME	If the storage was shut down, the time shut down is given here as a decimal.

ENTRIES	The number of transactions who entered the ENTER block was 68.
AVERAGE TIME/UNIT	The average time a transaction was in this block was 13.635 time units.
CURRENT STATUS	The storage is currently available, AVAIL
PERCENT AVAIL	The storage was available 100.0 percent of the time.
CAPACITY	The storage capacity was 2. This was specified by the STORAGE statement.
CURRENT CONTENTS	When the program ended the contents of the storage was 2.
MAXIMUM CONTENTS	The maximum number in the storage at any one time was 2.

SNA's Associated with Storages

The SNA's associated with the storage are given below:

SNA	meaning
S(name) or Sn	current storage content
R(name) or Rn	remaining storage content
SA(name) or SAn	average storage content
SC(name) or SCn	storage entry count
SM(name) or SMn	maximum storage count
SR(name) or SRn	utilization of storage in parts per thousand
ST(name) or STn	average holding time

Notice that the utilization of the storage is given by SR(name) and is expressed in parts per thousand. The original storage specification is not an SNA. If it is desired to use this in the program, it is necessary to use S(name)+R(name) which always adds up to the original storage specification.

Actually, it is not necessary to specify a storage when you have an ENTER block. In this case the processor sets aside 2,147,483,677 as the storage capacity. (The number is 2**31 -1). It may appear that you would never omit giving the storage capacity, but there are examples when one actually does omit this. For example, suppose you are modeling a hardware store. Customers arrive and immediately take a shopping cart. Suppose customers arrive every 30 ± 8 seconds. If you have 20 carts available, you would model this as

```
      STORAGE    S(CART),20    PROVIDE 20 CARTS
      GENERATE   30,4          CUSTOMERS ARRIVE
      ADVANCE    0             DUMMY BLOCK
      ENTER      CART          SELECT A CART
      .......
      .......
    rest of program
```

Notice that an ADVANCE block is used with zero time units. This is done so that if all the carts are taken, the transaction will not be held up in the GENERATE block. This insures that the GENERATE block will produce transactions according to the specified times of 30 +/ 4 seconds. The way the program is written, if a customer arrived and found the carts all taken, he or she would wait until one became free. This is a bit unrealistic. Suppose, instead, you wanted the customer to leave if no cart were available. This could be done as follows:

```
      STORAGE    S(CART),20
      GENERATE   30,4
      TRANSFER   BOTH,,OUT
      ENTER      CART
      .......
      .......
    other statements
      .......
OUT   TERMINATE
```

But, now suppose that you wanted to determine the maximum number of carts ever used for the simulation period. You want the program written so that arriving customers always were able to take a cart. One way would be to assign a very large storage for CART, such as:

```
    STORAGE S(CART),10000
```

However, we could just as easily omit the STORAGE statement. At the end of the program the maximum number of entries into CART are listed. (It is even possible to print out a table of the

statistical distribution of the carts used - this will be covered later.)

The LEAVE Block

Once a transaction uses parallel servers via the ENTER block it must eventually indicate that it is done and so release the server. The block to do this is the LEAVE block. Just as the QUEUE and DEPART blocks are `twins', so are the ENTER and LEAVE blocks. The form is similar to the ENTER block as it relates directly to it. This form is:

```
LEAVE   A
```

where A is the name (or number) of the parallel servers. As we shall see later, it can also be a variable.

Thus, you might have the following in a program:

```
STORAGE     S(JOE),4
........
ENTER       JOE
........
........
LEAVE       JOE
........
........
```

It may appear that there is no difference between a single facility and a storage with a capacity of 1. This is almost the case, but we shall see that when we study the concept of `preempting' that a facility can be preempted, whereas a storage cannot.

Example 8.2

A garage for inspecting a fleet of cars has three identical service areas. Cars arrive for inspection every 3 ± .3 minutes. These inspections take 8 ± 2 minutes. After leaving the inspection area, 70% are ready to return to service but the rest need further service which takes 4 ± 1.5 minutes. If this further service is needed, a single mechanic is assigned to do it. Simulate this system for one day.

The program to do this is as follows:

```
SIMULATE
```

```
        STORAGE    S(SPACE),3       PROVIDE SERVICE AREAS
        GENERATE   3,.3             CARS ARRIVE
        ADVANCE                     DUMMY BLOCK (ZERO CAN BE OMITTED)
        ENTER      SPACE            IS A SPACE AVAILABLE
        ADVANCE    8,2              INSPECTION TAKES PLACE
        LEAVE      SPACE            FREE INSPECTOR
        TRANSFER   .70,,OUT         70% RETURN TO SERVICE
        SEIZE      MECH             SINGLE MECHANIC
        ADVANCE    4,1.5            MECHANICS WORKS ON CAR
        RELEASE    MECH             FREE MECHANIC
OUT     TERMINATE                   CARS LEAVE
        GENERATE   480              SIMULATE FOR A DAY
        TERMINATE  1
        START      1
        END
```

Example 8.3

A hardware store consists of four aisles and a single checkout counter. Shoppers arrive with interarrival time of 82.5 ±26.4 seconds. After arriving each customer who plans to shop in any one or more of the aisles takes a shopping cart. However, 12% of the customer simply go to the checkout counter where various items are for sale. These people do not take a shopping cart. The rest shop down each aisle as follows:

aisle	prob of going down	time required to travel
1	.80	125 ± 70
2	.75	140 ± 40
3	.85	150 ± 65
4	.90	175 ± 70

When a shopper is finished he or she will join the queue in front of the counter until each is checked out. The time to check out is 45 ± 12 seconds for those who shop in the aisles and 35 ± 12 for those who go directly to the checkout counter.

The store owner is concerned that he does not have enough shopping carts. Customers who arrive and find none available tend to leave and shop elsewhere. In addition, the single person who

works at the checkout counter is complaining that she is working too hard and is threatening to contact her union. Union regulations forbid a person from working more than 85% of the time. Determine how many shopping carts the store should have.

Solution

The program to solve this problem is given below:

```
      SIMULATE
      STORAGE    S(CARTS),1000     PROVIDE 1000 CARTS
      GENERATE   85.2,26.4         CUSTOMERS ARRIVE
      TRANSFER   .12,,COUNTER      12% GO TO COUNTER
      ENTER      CARTS             REST TAKE A CART
      TRANSFER   .2,,AISLE2        80% GO TO AISLE 1
      ADVANCE    125,70            SHOP IN AISLE 1
AISLE2 TRANSFER  .25,,AISLE3       75% GO TO AISLE 2
      ADVANCE    140,40            SHOP IN AISLE 2
AISLE3 TRANSFER  .15,,AISLE4       85% GO TO AISLE 3
      ADVANCE    150,65            SHOP IN AISLE 3
AISLE4 TRANSFER  .10,,CHECK        90% GO TO AISLE 3
      ADVANCE    175,70            SHOP IN AISLE 3
CHECK QUEUE      LINE              STAND IN LINE
      SEIZE      WORKER            READY TO CHECK OUT
      DEPART     LINE              LEAVE THE QUEUE
      ADVANCE    45,20             CHECK OUT
      RELEASE    WORKER            FREE THE CHECK OUT GIRL
      LEAVE      CARTS             GET RID OF CART
      TERMINATE                    LEAVE THE STORE
COUNTER QUEUE    LINE              STAND IN LINE
      SEIZE      WORKER            READY TO CHECK OUT
      DEPART     LINE              LEAVE THE QUEUE
      ADVANCE    35,12             CHECK OUT
      RELEASE    WORKER            FREE THE CHECK OUT GIRL
      TERMINATE                    LEAVE THE STORE
      GENERATE   28000             SIMULATE FOR 1 DAY
      TERMINATE  1                 TIMER TRANSACTION
      START      20                SIMULATE FOR 20 DAYS
      END
```

Selected portions from the output are as follows:

```
RELATIVE CLOCK: 5.6000E+05 ABSOLUTE CLOCK: 5.6000E+05

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               6539   11     1        5147   21              792
2               6539   CHECK           5743   22              792
3               5747   13              5743   23              792
4               5747   14              5743   24              792
5               4573   15              5743   25              20
AISLE2          5747   16              5743   26              20
7               4358   17              5743
AISLE3          5747   18              5743
9      3        4890   COUNTER         792
AISLE4          5744   20              792

          --AVG-UTIL-DURING--

FACILITY   TOTAL    ENTRIES    AVERAGE
           TIME               TIME/XACT
WORKER     0.510     6535       43.688

          --AVG-UTIL-DURING--

STORAGE    TOTAL    ENTRIES    AVERAGE   CURRENT   MAXIMUM
                               CONTENTS  CONTENTS  CONTENTS
CARTS      0.006     5747       5.633       4        10

QUEUE     MAXIMUM   AVERAGE   TOTAL     ZERO      AVERAGE     $AVERAGE
          CONTENTS  CONTENTS  ENTRIES   ENTRIES   TIME/UNIT   TIME/UNIT
LINE         4       0.173     6535      3711      14.864      34.396
```

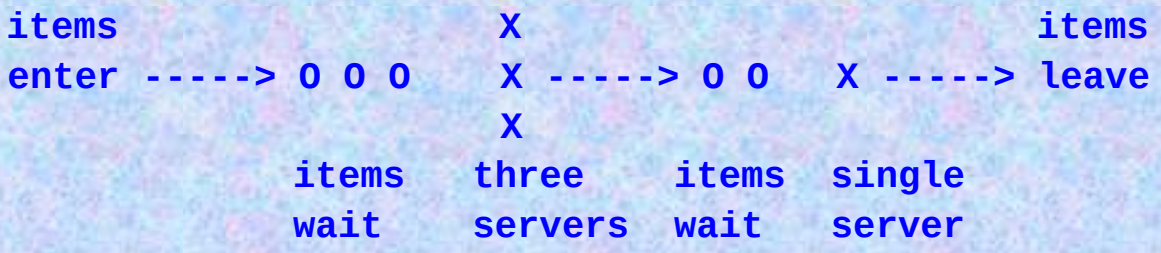
The number of customers arriving at the store in the 20 days was 6539. Of these 5747 took a cart and shopped in the aisles. 792 went directly to the check out counter. The checkout girl was busy only 51.0% of the time so she has no complaint of being overworked.

The maximum number of carts ever in use was 10. At this time, it would be instructive to know how many times this happened. Later it will be shown how to make statistical distributions to provide us with this but, for the present, we do not have this data. However, it would seem that

providing something like 12 carts should be sufficient. This would take into account the maximum number obtained in the simulation as well as provide a safety factor of 2 extra carts.

Exercise 8.

1.



Consider the above diagram. Items enter the system every 115 ± 30 seconds. They wait in a queue until one of three identical servers is ready. Service takes 335 ± 60 seconds. After this service, they move along and join another queue whilst they wait for a single server to perform another service. This server works in 110 ± 25 seconds. Design the model to measure the waiting line behavior ahead of the two places where service is performed. Assume first that there is unlimited space between the three servers and the single server.

2. In Exercise 1, suppose that there can only be 1 item in the waiting area before the second server. Modify the program to take this into account.

3. In Exercise 1, suppose further that not only can there by only 1 person in the waiting area but that the first three servers cannot begin work on another item until the finished item is placed on the waiting area for the single server. Modify the program further.

4. Ships of two types arrive at a harbor, where they unload their cargo. There are two tugs which service the harbor. Type 1 ships are small and so they need only one tug to both dock and undock. Type two ships are large and they need 2 tugs to both dock and undock. Because of their size differences the ships dock at different berths and have different loading/unloading time requirements. Data is as follows:

	ship type	
	1	**2**
interarrival time	130 ± 30	390 ± 60
docking time	30 ± 7	45 ± 12
number of berths	6	3

unloading time	720 ± 120	1080 ± 240
undocking time	20 ± 5	35 ± 10

A ship cannot use a tug unless a berth is free for it. A type 2 ship has priority over a type 1 ship. Model the harbor. Suppose it cost $350/hr for a ship of type 1 to wait for a berth and $500/hr for a ship of type 2 to wait, and the cost of a tug is $250/day. Would you recommend the addition of a third tug? (Its initial cost is immaterial). What is the optimum number of tug boats to have?

5. The following is one of the most remarkable examples of the power of the GPSS language. The exercise is adapted from one in Schriber's 1974 textbook.

A mine is in full operation. Trucks haul the ore and periodically break down. When they break down, they are taken to the repair area to be fixed. Once trucks are fixed they are returned to the mine. They are then classified as spares as the mine has more trucks available than are needed at any one time in the mine.

The mine needs 30 trucks for optimum production (this was determined from a previous simulation study). The number of repair facilities can vary but you estimate that either 3, 4, or 5 should be correct. The cost of each is $77/8 hr shift. Each spare truck you have costs you $35/shift no matter what it does. Thus, if you happen to have 33 trucks and 5 repair facilities, the cost/8 hours is:

5 x 77 + 3 x 35 = $490

For 3 repair facilities and 34 trucks the costs are:

3 x 77 + 4 x 35 = $371

It is possible to make a table of costs for various combinations of repair facilities and number of spare trucks. Each time you do not have 30 trucks in the mine it costs you $12/hr or $96/shift. Trucks mean time between breakdown of a truck is:

230 ± 40 hours

The mean time for repairs of a truck is:

25 ± 9 hours.

Determine the optimum number of both trucks and repair facilities to have.

6. A manufacturing system consists of two waiting lines and two servers. Only 4 units can wait at station 1 and 2 at station 2. If another unit comes along when the waiting space a station 1 is full, it leaves and a penalty is incurred. If a unit from station 1 is finished but 2 units are waiting at station 2, the unit remains in station 1 until space is free. While a unit is in station 1 no other units can use it.

```
=======> o o o o ======> station 1 =====> o o station 2 ======>
   |
   ====> units turned away
```

Arrivals are exponentially distributed with a mean of 0.4 time units. Service times are exponentially distributed at both stations with means of 0.25 and 0.5 respectively. Model this system to see how efficient it is. Simulate for 500 time units.

7. In Exercise 6, change the station working times to have means of .35 for station 1 and .40 for station 2. Note that their sum is still 0.75 as it was before. Is the system improved?

8. Repeat exercise 6 with the original data but now with waiting space for the stations allocated as 3 and 3.

9. Repeat exercise 8 with the but now with waiting space for the stations allocated as 3 and 3.

Solutions

1.
```
STORAGE    S(SERV1),3
GENERATE   115,30
QUEUE      LINE1
ENTER      SERV1
DEPART     LINE1
ADVANCE    335,60
LEAVE      SERV1
QUEUE      LINE2
SEIZE      SERV2
DEPART     LINE2
ADVANCE    110,25
RELEASE    SERV2
TERMINATE
GENERATE   280000            10 DAYS SIMULATION
TERMINATE  1
```

```
        START       1
        END

   6.   SIMULATE
        STORAGE     S(ONE),4/S(TWO),2   ROOM FOR 4 AND 2
        GENERATE    RVEXPO(1,.4)        UNITS ARRIVE
        TRANSFER    BOTH,,AWAY          IS THEIR ROOM?
        ENTER       ONE                 YES, JOIN FIRST QUEUE
        SEIZE       MACH1               DO FIRST SERVICE
        LEAVE       ONE                 LEAVE THE QUEUE
        ADVANCE     RVEXPO(1,.35)       PERFORM SERVICE
        ENTER       TWO                 IS THERE A SEAT FREE
        RELEASE     MACH1               YES, FREE MACHINE 1
        SEIZE       MACH2               USE MACHINE 2
        LEAVE       TWO                 LEAVE SEAT 2
        ADVANCE     RVEXPO(1,.4))       PERFORM SERVICE
        RELEASE     MACH2               FREE MACHINE 2
        TERMINATE                       PART LEAVES
AWAY    TERMINATE                       SORRY, SYSTEM FULL
        GENERATE    500                 500 TIME UNITS
        TERMINATE   1                   END OF PROGRAM
        START       1
        END
```

Selected output is as follows:

```
RELATIVE CLOCK: 500.0000 ABSOLUTE CLOCK: 500.0000

BLOCK  CURRENT  TOTAL   BLOCK   CURRENT  TOTAL
1               1172    11               959
2               1172    12               959
3               960     13               959
4               960     AWAY             212
5               960     15               1
6       1       960     16               1
7               959
8               959
9               959
10              959
```

```
          --AVG-UTIL-DURING--

FACILITY   TOTAL    ENTRIES   AVERAGE
           TIME               TIME/XACT
MACH1      0.877      960      0.457
MACH2      0.781      959      0.407

          --AVG-UTIL-DURING--

STORAGE    TOTAL  ENTRIES  AVERAGE  CAPACITY  AVERAGE  CURRENT  MAXIMUM
           TIME            TIME/UNIT          CONTENTS CONTENTS
CONTENTS
ONE        0.472  960       0.983       4       1.888       0        4
TWO        0.487  959       0.508       2       0.974       0        2
```

SPARE TRUCK PROBLEM

The following is one of the most remarkable examples of the power of the GPSS language. The exercise is adapted from one in Schriber's 1974 textbook.

A mine is in full operation. Trucks haul the ore and periodically break down. When they break down, they are taken to the repair area to be fixed. Once trucks are fixed they are returned to the mine. They are then classified as spares as the mine has more trucks available than are needed at any one time in the mine.

The mine needs 30 trucks for optimum production (this was determined from a previous simulation study). The number of repair facilities can vary but you estimate that either 3, 4, or 5 should be correct. The cost of each is $77/8 hr shift. Each spare truck you have costs you $35/shift no matter what it does. Thus, if you happen to have 33 trucks and 5 repair facilities, the cost/8 hours is:

5 x 77 + 3 x 35 = $490

For 3 repair facilities and 34 trucks the costs are:

3 x 77 + 4 x 35 = $371

It is possible to make a table of costs for various combinations of repair facilities and number of spare trucks. Each time you do not have 30 trucks in the mine it costs you $12/hr or $96/shift. Trucks mean time between breakdown of a truck is:

230 ± 40 hours

The mean time for repairs of a truck is:

25 ± 9 hours.

Determine the optimum number of both trucks and repair facilities to have.

Exercise

A hardware store consists of four aisles and a single checkout counter. Shoppers arrive in a Poisson pattern with a mean interarrival time of 82.5 seconds. After arriving each customer who plans to shop in down any one of the aisles takes a shopping cart. However, 12% of the customer simply go to the checkout counter where various items are for sale. These people do not take a shopping cart. The rest shop down each aisle as follows:

aisle	prob of going down	time required to travel
1	.80	125 ± 70
2	.75	140 ± 40
3	.85	150 ± 65
4	.90	175 ± 70

When a shopper is finished he or she will queue in front of the counter until each is checked out. The time to check out is found to be 45 ± 12 seconds for those who shop in the aisles and 20 ± 8 for those who go directly to the checkout counter.

The store owner is concerned that he does not have enough shopping carts. Customers who arrive and find none available tend to leave and shop elsewhere. In addition, the single person who works at the checkout counter is complaining that she is working too hard and is threatening to contact her union. Union regulations forbid a person from working more than 85% of the time. Determine how many shopping carts the store should have.

Solution

The program to solve this problem is given below:

```
      SIMULATE
      STORAGE    S(CARTS),1000      PROVIDE 1000 CARTS
      GENERATE   RVEXPO(1,82.5)     CUSTOMERS ARRIVE
      TRANSFER   .12,,COUNTER       12% GO TO COUNTER
      ENTER      CARTS              REST TAKE A CART
      TRANSFER   .2,,AISLE2         80% GO TO AISLE 1
      ADVANCE    125,70             SHOP IN AISLE 1
AISLE2 TRANSFER  .25,,AISLE3        75% GO TO AISLE 2
      ADVANCE    140,40             SHOP IN AISLE 2
AISLE3 TRANSFER  .15,,AISLE4        85% GO TO AISLE 3
      ADVANCE    150,65             SHOP IN AISLE 3
AISLE4 TRANSFER  .10,,CHECK         90% GO TO AILSE 3
      ADVANCE    175,70             SHOP IN AISLE 3
CHECK QUEUE      LINE               STAND IN LINE
      SEIZE      WORKER             READY TO CHECK OUT
      DEPART     LINE               LEAVE THE QUEUE
      ADVANCE    35,12              CHECK OUT
      RELEASE    WORKER             FREE THE CHECK OUT GIRL
      LEAVE      CARTS              GET RID OF CART
      TERMINATE  LEAVE              THE STORE
COUNTER QUEUE    LINE               STAND IN LINE
      SEIZE      WORKER             READY TO CHECK OUT
      DEPART     LINE               LEAVE THE QUEUE
      ADVANCE    45,20              CHECK OUT
      RELEASE    WORKER             FREE THE CHECK OUT GIRL
      TERMINATE                     LEAVE THE STORE
      GENERATE   3600*8*20          SIMULATE FOR 20 DAYS
      TERMINATE  1                  TIMER TRANSACTION
      START      1
      END
```

Selected portions from the output are as follows:

```
RELATIVE CLOCK: 5.7600E+05 ABSOLUTE CLOCK: 5.7600E+05

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               7025   11     2        5576   21              834
2               7025   CHECK           6180   22              834
3               6191   13              6180   23              834
```

```
4              6191   14              6180   24              834
5      4       4937   15              6180   25              1
AISLE2         6187   16              6180   26              1
7      3       4645   17              6180
AISLE3         6184   18              6180
9      2       5271   COUNTER          834
AISLE4         6182   20               834

        --AVG-UTIL-DURING--

FACILITY   TOTAL   ENTRIES   AVERAGE
           TIME             TIME/XACT
WORKER     0.441    7014      36.184

        --AVG-UTIL-DURING--

STORAGE   TOTAL   ENTRIES   AVERAGE   AVERAGE   CURRENT   MAXIMUM
                            TIME/UNIT CONTENTS  CONTENTS  CONTENTS
CARTS     0.006    6191      540.518   5.810       11        16

QUEUE   MAXIMUM   AVERAGE   TOTAL   ZERO    PERCENT   AVERAGE
$AVERAGE
        CONTENTS  CONTENTS  ENTRIES ENTRIES  ZEROS   TIME/UNIT
TIME/UNIT
LINE      7         0.183     7014    3896     55.5    15.023    33.794
```

The number of customers arriving at the store in the 20 days was 7025. Of these 6191 took a cart and shopped in the aisles. 834 went directly to the check out counter. The checkout girl was busy only 44.1% of the time so she has no complaint of being overworked.

The maximum number of carts ever in use was 16. At this time, it would be instructive to know how many times this happened. Later it will be shown how to make statistical distributions to provide us with this but, for the present, we do not have this data. However, it would seem that providing something like 18 carts should be sufficient. This would take into account the maximum number obtained in the simulation as well as provide a safety factor of 2 extra carts.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 9.
The CLEAR, RESET and RMULT statements

The CLEAR Statement

A GPSS/H program is first complied. During this phase transactions are poised on the CEC to be moved by the processor when the actual program begins. There will be one transaction from each GENERATE block poised to be moved. As soon as this transaction is moved another one is scheduled to be moved.

When a program encounters the first START statement it begins execution. If there are other statements after the START statement, the processor will execute them after the initial program is executed. As far as understanding how the running of the program takes place, you can ignore any statements after the first START statement. For example, suppose you had other START statements one after the other such as:

```
GENERATE   10,5
........
........
........
GENERATE   480
TERMINATE  1
START      1
START      1
START      1
END
```

The effect of the above is to run the program three times (just as if you had START 3. But now there will be additional output from the three times, 480, 960 and 1440. Actually, it is possible to have the START statement with operands as follows:

```
START  A,NP,B,1
```

NP must be the letters "NP". This stand for "no printout" and no output file is created.

B is called the "Snap" interval. Every time the A counter is decremented by this amount, there will be output. If the last operand is used, it must be the number 1. When used, the processor

outputS the various chains (Current Events Chain, Future Events Chin, etc.). As an example, if you had,

```
    START 100,,10
```

You would get output for when the counter was 90, 80, 70, etc.

When you have multiple START statements, the program keeps running from the point where the last program left off. The transactions remain where they were and the program keeps going. All statistics are updated from where they were.

Often you will want to run the program multiple times with different values in the blocks. There are several ways to do this. One is as follows:

Give the blocks you want to change different labels, i. e., suppose you want to change a GENERATE block give it a label such as:

```
KEY1  GENERATE 12,6
```

Run the program in the normal manner, i. e.:

```
      GENERATE   480
      TERMINATE  1
      START      1
```

After the START statement simply place the replacement block and have another START statement:

```
      START      1
KEY1  GENERATE   11,6
      START      1
```

The program will run initially with the block GENERATE 12,6 and produce an output report. It will then run again with the GENERATE block replaced by the block GENERATE 11,6. The only problem with this is that the program will continue to run with the previous program's statistics and transactions already in the system. This is probably not what you want. The next statement will show how to run the program a second time with the previous statistics zeroed out.

The CLEAR Statement

The CLEAR statement allows a program to be run multiple time with the statistics of the system set to zero. This is simply the statement CLEAR. This will clear all the transactions out as well as the previous statistics. Thus,

```
      START      1
KEY   GENERATE   11,6
      CLEAR
      START      1
```

will run a program two times, the second time with GENERATE 12,6 and the second time with GENERATE 11,6. When you do this, there will be a warning message that you have named the block multiple times - you ignore it as you intentionally did this. Whenever you label a block in GPSS and do not reference it in the program, there will be a warning message.

Since the CLEAR statement clears all statistics, there will be a time when you will want only selected items cleared. This can be done by putting the items you do not want cleared after the CLEAR statement:

```
CLEAR   item1, item2,...
```

The effect of the above CLEAR statement is to clear all items except $item_1$, $item_2$,....

The RESET Statement

There will be times when you do not want to clear out the transactions (or the various chains). For example, suppose you want to run a program for an hour (when the model is empty), discard the statistics but not the positions of the transactions. You do the following:

```
      START     1,NP        Simulate for an hour
      RESET                 RESET the statistics
      START     100         Simulate for 100 hours
```

The above is often done when the first hour's (or so) statistics are to be discarded due to instability in the system.

Note: Whenever you use CLEAR or RESET you do NOT change the position of the random number generator. This is a pseudo-random number generate which means that the numbers can be repeated. This is important in designing comparison of models. How, then, can you restart the random number stream at the same place? This is done by the RMULT statement.

The RMULT Statement

The RMULT statement followed by any number in the range ± 2,147,483,646 to be used in the operand starts the random number stream at a particular point. Thus, if you had:

```
        SIMULATE
        RMULT       54321
        .....
        .....
        .....
        START       1
        CLEAR
        RMULT       54321
KEY     GENERATE    10,6
        START       1
```

The second time the program is run, the same random numbers would be used.

GPSS/H uses any number of random number streams. Up until now we have been using only the first, by default. When you are using multiple streams and want to reset each, the RMULT statement is:

```
        RMULT  1234,6543,,99,,231,999
```

Here random number streams 1, 2, 4, 6 and 7 are reset.

Encountering a group of CLEAR, RESET and RMULT statement at the end of a program may seem confusing at first. But, remember that these are control statements and NOT part of the program blocks. Always remember that the program starts execution after each START statement and ignores the remaining statements until execution ends.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 10
FUNCTIONS

Functions in GPSS/H

Up to now the only statistical distribution used was the uniform distribution. In GPSS, it is possible to sample from any statistical distribution. This is very important since to model most things that are moving through a system, it will be necessary to sample from many different distributions. Some times these may be the exponential or Poisson distribution, at times it may be the normal or Gaussian distribution, or perhaps a distribution that is unique to the model. There are, in fact, many times when we will refer to a function to return a value to be used in the simulation. There are several types of functions in GPSS. One will be given in this chapter and one in the next.

Discrete Functions

A function must first be defined and then referenced when a value is to be obtained from it. The first form of a function to be discussed here is known as a discrete function. This is so called because when referenced, it will take on only one of several set values. These possible values are specified when the function is defined. For example, an order for goods may take 4, 5, 6, or 7 weeks to arrive. The number of people away from a class due to illness might be from 0 to 12.

The way to reference discrete functions in GPSS is given next. First, the function must be defined by means of the FUNCTION statement. This has to be done before it is referenced. The first line of the function definition might be as follows:

```
(label) FUNCTION RN1,Dn
```

where (label) is the name or number of the function. This will be used to reference the function. RN1 is a built in standard numerical attribute. When it is reference and used in a function, a number from 0.000000 to 0.999999 is returned. Actually, any standard numerical can be used in place of RN1. Examples of this are given later in the chapter. The number 1 in RN1 is arbitrarily chosen. Any other number could have been selected. This is the number of the random number stream to be sampled from. Dn is the letter "D" followed by an integer. The number D gives the number of pairs of values to sample from.

Whenever RNj (or RN(j)) is referenced a random number is returned. If the reference is in

connection with a function, the number returned is between .000000 and .999999 (from 0 to 1 but never 1.000000). When RNj is used in other situations, the number returned is between 000 and 999. For example,

```
FIRST FUNCTION RN1,D2
```

would be a function named FIRST. It can take on two values.

```
TOP FUNCTION RN3,D5
```

is the function TOP which will take one of five values.

```
FIVE FUNCTION RN6,D15
```

refers to the function FIVE which can have one of 15 values. It is possible to have labels for functions that are numbers such as

```
6 FUNCTION RN1,D5
```

On the line or lines below the function statement are the Dn possible values which the function might assume. These are in pairs separated by a comma and normally in ascending order although they need not necessarily be. The pairs are themselves separated by a slash "/". These values can start in position 1 (this is one of the few times in GPSS that anything can be in position 1) and go up to and including position 72. The pairs can occupy more than one line, if necessary. If so, the slash is omitted from the end of the previous line. The first number of the pair refers to the GPSS generated random number.

The way a discrete function works is as follows:

When a function having RN1 as an operand is referenced, a random number is obtained from the random number stream referenced. In this case the first random number stream is used because of the 1 in RN1. RN5 would reference random number stream 5. There is no limit to the number of random number streams in GPSS/H, although most references will be to RN1 for convenience.

This random number returned is then used to obtain a value to be returned. Thus,

```
SALLY FUNCTION RN1,D3
   .1,5/.6,8/1,10
```

will return either 5, 8 or 10 and no other values. If the random number obtained from the random number stream is between .000000 and .100000, the number returned is 5; if the number is from

.1000001 to .600000, the number returned is 8; if the number is from .600001 to .999999 the number returned is 10. This is better seen as follows:

random number	value returned
.000000 <= RN <= .100000	5
.100001 <= RN <= .600000	8
.600001 <= RN <= .999999	10

This is known as Monte Carlo sampling. Simulations done using this form of sampling is Monte Carlo simulation.

Referencing Functions in GPSS/H

Functions are referenced in GPSS/H by putting the letter FN followed by left parenthesis "(", the name of the function and the right parenthesis ")". (In the case of using a number for a function, reference can be simply by FNj where j is the number).

Some examples are as follows:

```
a) ADVANCE   FN(TIME)
b) GENERATE  FN(SPEED)
c) GENERATE  ,,,4,FN(AMOUNT)
d) GENERATE  100,25,FN(TIMEIN)
```

In a), the transaction will be placed on the FEC for a time given by reference to the function TIME.

In b), a transaction will be generated according to the time given by the function SPEED. This means that during compiling a transaction is scheduled to leave the GENERATE block. As soon as this transaction leave, another is scheduled to leave. The times to leave are given by reference to the function SPEED.

In c) will generate 4 transactions at time 0. The priority of each will be given by the function AMOUNT.

In d) will generate transaction every 100 ± 25 time units. The first transaction will enter the system at a time given by reference to the function TIMEIN.

The number returned when a function is referenced need not be an integer. Thus, it would be possible to have:

```
TEST6 FUNCTION RN3,D4
.25,3.5/.44,5.1/.7,7.8/1,9.89
```

Here the returned values would be either 3.5, 5.1, 7.8 or 9.89. If you had the following:

```
TEST7 FUNCTION RN1,D3
.1,4/.5,6/.8,9
```

and a random number greater than .8 was returned when RN1 was sampled, the value of the function is 9.

Example 10.1

Suppose a shopper takes either 3, 4 or 5 minutes to load a shopping cart. 30 % of the time it takes 3 minutes, 30 % of the time it takes 4 minutes and the remaining 40% of the time it takes 5 minutes. To use a GPSS/H function, it is necessary to make a cumulative probability distribution. Table 10.1 illustrates this distribution.

Table 10.1 Cumulative Probabilities for Shopper

time (min)	prob.	cum. prob.
3	.30	.30
4	.30	.60
5	.40	1.00

Since 30 % of the time it takes the shopper 3 minutes to load the cart, we would like to have the GPSS function return a time of 3 minutes also 30 % of the times. This is done by assigning the value of the random number generated by the GPSS processor to the corresponding probability. This assignment is as shown in Table 10.2.

Table 10.2 Random Number Assignment.

time	cum. prob.	random number
3	.30	.000000 to .300000
4	.60	.300001 to .600000
5	1.00	.600000 to .999999

If the random number is between .000000 and .300000 the time is taken as 3; if the random number is from .300001 to .600000, the time is 4 and if the random number is from .600000 to .999999, the value is 5. Notice that by using cumulative probability distributions, the correspondence between the random number and a time is unique, i.,e., for each random number there is one and only one value that corresponds. Notice, also, that there is a very slight error since the random number is never equal to 1.000000. This is so small that it is not significant.

Returning to the original problem, we would write the function as:

```
SHOPR FUNCTION RN1,D3
.3,3/.6,4/1,5
```

Notice that there is no decimal after the 1 in the pair 1,5. The decimal is optional here. There is no slash at the beginning or end of the pairs of numbers. It would have been all right to also have:

```
SHOPR FUNCTION RN1,D3
.3,3/.6,4
1,5
```

or even to have each pair of values on a separate line. Consider the following

```
TIME FUNCTION RN1,D4
.25,8/.50,9/.9,10/1,11
```

Later in the program if you had,

```
ADVANCE FN(TIME)
```

the transaction would be put on the future events chain for a time of 8, 9, 10, or 11 units. It will be 8 for 25 % of the time; 9 for 25 % of the time; 10 for 40 % of the time and 11 for the remaining 10 % of the time. If we had,

```
ADVANCE FN(TIME),5
```

the transaction would be on the future events chain for one of the following times:

8 ± 5, 9 ± 5, 10 ± 5 or 11 +/ 5 time units.

Caution in Using Functions with ADVANCE and GENERATE Blocks

If you had the following functions

```
ONE FUNCTION RN1,D3
.3,10/.6,15/1,20
TWO FUNCTION RN1,D2
.4,2/1,3
```

and you used

```
ADVANCE FN(ONE),FN(TWO)
```

If FN(ONE) was 15 and FN(TWO) was 3, the transaction would be placed on the future events chain for 45 time units, not 15 ± 3 time units. Later, we shall see how to handle the example used here.

Example 10.2

Customers arrive at a car park every 100 ± 23 seconds. It takes them 13 ± 5 seconds to park and 12 ± 2 seconds to walk to the store. Shopping time varies as follows:

Table 10.4 Shopping time in store.

time (seconds)	rel. freq.	cum. freq.
50	.12	.12
55	.25	.37
65	.27	.64
75	.22	.86
90	.14	1.00

The store is small and has only one checkout counter. The time to checkout is as follows:

Table 10.5 Checkout times for shoppers.

time (seconds)	rel. freq.	cum. freq.
80	.25	.25
90	.20	.45
100	.30	.75
120	.25	1.00

10% of the shoppers make no purchase and leave the store. The time to return to their cars is 20 ± 8 seconds and the time to leave the car park is 8 ± 4 seconds. Simulate for 1000 people coming and then leaving the store.

The program to do the simulation is:

```
      SIMULATE
SHOP  FUNCTION   RN1,D5
.12,50/.37,55/.64,65/.86,75/1,90
CHKOT FUNCTION   RN1,D4
.25,80/.45,90/.75,100/1,120
      GENERATE   100,23          CUSTOMERS ARRIVE
      ADVANCE    13,5            PARK CAR
      ADVANCE    12,2            WALK TO STORE
      ADVANCE    FN(SHOP)        SHOP
      TRANSFER   .1,,OUT         10% MAKE NO PURCHASE
      QUEUE      LINE            REST QUEUE AT CHECKOUT
      SEIZE      GIRL            USE CHECKOUT GIRL
      DEPART     LINE            LEAVE QUEUE
      ADVANCE    FN(CHKOT)       CHECKOUT
      RELEASE    GIRL            FREE CHECKOUT GIRL
OUT   ADVANCE    20,8            RETURN TO CARS
      ADVANCE    8,4             LEAVE CAR PARK
      TERMINATE  1
      START      1000
      END
```

While this example illustrates the use of the FUNCTION statement, it is not realistic. The most

glaring deficiency is the fact that the various shopping and checkout times are given by discrete values. Certain times such as 51 seconds for shopping, and 97 seconds for checking out are not considered. They could have been included by making the FUNCTION statement much longer. This is tedious and not necessary, as we shall learn in the next chapter However, the following points should also be considered in building an actual model of a grocery store.

1. Shoppers do not arrive during the day according to the same statistical distribution. At certain times (between 4.30 p. m. and closing, for example), there are more shoppers than during other times. A more nearly accurate function should reflect this.

2. The parking lot may hold only a finite number of cars. If another car comes when the lot is full, it may leave. This can be programmed by means of the STORAGE statement and the ENTER block.

3. It might be better to have the checkout time vary depending on the number of items purchased.

4. Some shoppers may not join the queue if it is too long, but will prefer to continue shopping.

5. If the queue reached a certain length, the one person working may be able to call for another helper.

6. The time to walk from the parking lot to the store (and subsequently return) should vary with the number of cars in the lot. The more cars in the lot, the farther away the next car that arrives will have to park.

As we learn more GPSS, it will be possible to incorporate the above changes into the model.

Continuous Functions

A continuous function is defined an referenced in much the same manner as a discrete one. In the case of discrete functions, only a finite number of values are returned, which is specified by the Dn. For continuous functions a value is returned that can be considered as being a decimal within a specified range, i. e., if the range is from 4 to 7, any possible value from 4.000000 to 7.000000 can be returned.

The pairs giving the ranges must be ordered and the number of these pairs is given by Cn. One form of a continuous function might have the first line as:

```
TEST1  FUNCTION  RN1,C5
```

This means that the function named TEST1 will use random number stream 1 to return a random number. The value of the function will be determined by sampling from 5 pairs of numbers. The ordered pairs of number that come on the line(s) after the function line specify the intervals where the value of the function is to be obtained. This is done by a linear interpolation between the pairs. For example, the function defined as follows:

```
TEST2  FUNCTION  RN1,C3
0,1/.7,4/1,5
```

will return values in the intervals (0,1) to (.7,4) and the intervals (.7,4) and (.999999,5). Suppose when the function is referenced, the random number returned is .3000000. The value of the function will be 2.285714 which is obtained by a linear interpolation from (0,1) and (.7,4). Consider the function:

```
TEST3  FUNCTION  RN1,C2
0,1/1,5
```

This will return a value between 1.000000 and 4.999999 but not 5.000000 because the random number is from 0.000000 to .999999.

Use of Functions with any SNA

It is possible to have any SNA in a function definition. Thus, one could have the following:

```
TIMES  FUNCTION  Q(WAIT),D5
0,10/1,8/2,6/3,5/4,4
FIRST  FUNCTION  FR(MACH1),D3
100,5.5/500 ,6.6/923,7.6
SECOND FUNCTION  SR(BOOTH),D2
500,20/999,30
```

The function TIMES will return a value of 10, 8, 6 5 or 4 depending on the current length of the queue WAIT. The function FIRST will return a value of 5.5, 6.6 or 7.6 depending on the fractional utilization of the facility FIRST. The function SECOND will return a value of either 20 or 30 depending on the utilization of the storage BOOTH. an example of the use of such a function reference might be a barber who cuts hair at a rate that depends on the number of customers waiting in the shop. As the queue increases, the barber will increase his cutting rate. For example, suppose the normal time is 10 minutes if the queue is zero. If one customer is waiting, the time is 9 minutes, if two are waiting the time is 8.5 minutes and if three or more are waiting the time is 8 minutes. The program might have the following lines of code:

```
CUTTING  FUNCTION   Q(WAIT),D4
0,10/1,9/2,8.5/3,8
         .........
         .........
         ADVANCE    FN(CUTTING)
```

Other Forms of Functions

There are several other forms of functions that are used in GPSS/H. These are as follows:

1. List functions. Many times a function will have as the first number in is referencing pairs, the integers 1, 2, 3, ..., N. In this case it is called a "list" function and this is specified by the letter L as follows:

```
EXAMPLE  FUNCTION  Q(WAIT),L3
1,1/2,4/3,5
```

It would be wrong to have:

```
EXAMPLE  FUNCTION  Q(WAIT),L4
0,1/1,1/2,4/3,5
```

A list function is preferable to use if it should occur. Not only does it take less time to execute, but if the value of the SNA is outside the range 1 to N, an execution error occurs. This can be of use in de-bugging a program. Thus, in the example cited here, if the queue at WAIT was zero, the program would return an error. As we learn more programming code, there will be more examples of list functions.

2. Entity valued functions. It is possible to have functions that reference other functions. These are known as attribute valued functions and are specified by the letter E. For example,

```
SPEED  FUNCTION  RN1,E3
.25,FN(ONE)/.6,FN(TWO)/1,6
```

If the value of the random number is less than or equal to .25, the value returned is obtained by reference to the function ONE; if the value of RN1 is from .25 to .6 the value is obtained by reference to function TWO; otherwise the value is 6.

If an attribute function is also a list function, the letter E is replaced by the letter M. Again, there will be further examples of these functions as more GPSS/H code is introduced.

Exercises

1. Refer to Example 10.1. Suppose the parking lot only held 10 cars and so any other cars that came would be turned away each week. Assume the store is in operation for 5 day/week for 10 hours per day.

2. Build a function to return the following times:

time	prob.
5	.10
6	.20
7	.40
8	.30

3. A study was made of the time it takes Joe to give a haircut. 155 customers were timed and the following data was obtained:

time (minutes)	number of people
10	87
11	35
12	20
13	13

Show what the FUNCTION statement would be like to be used in the ADVANCE block to simulate Joe giving a haircut.

4. Table 10.6 gives the interarrival rate of customers coming into a repair shop. Repairs are done immediately and the customers will wait until they are finished. There is only one repairman.

Table 10.6 Interarrival rate for people at a store.

time between arrivals (min.)	rel. freq.
10	.10
11	.25
12	.35
13	.20
14	.08
15	.02

There are only two chairs for people to wait in and so if they are both taken, no more customers will enter the shop (this give a maximum number of three people in the store, not counting the person who gives the service). Service time varies according to the data in Table 10.7.

Table 10.7 Service time for customers in repair shop

time for service (minutes)	freq.
31	.25
32	.22
33	.28
34	.18
35	.07

Determine how many customers are turned away in a typical day.

5. A widget manufacturing process is as follows:

a) A worker takes a partially finished widget from a large pile. This can be considered as infinite.
b) He then finishes the assembling of it.
c) He takes it to a painting machine for final painting. There is only one of these painting machines and the person who assembled the widget is responsible for painting it.
d) He finishes the painting, puts the widget on a conveyor belt and return to his original work station to begin assembling another widget.

The various times associated with the above are:

a) assemble:

time (min.)	prob.	cum. prob.
20	.05	.05
21	.12	.17
22	.18	.35
23	.21	.56
24	.25	.81
25	.15	.96
26	.04	1.00

b) take to paint area: 1 ± .4 min.

c) paint a widget:

time (min.)	prob.	cum. prob.
6	.10	.10
7	.25	.35
8	.35	.70
9	.23	.93
10	.07	1.00

d) return to work station: 1 + .4 min.

Each widget earns the company a profit of $12.75. A worker is paid $9.50/hour. The fixed costs of the operation is $75/day. Determine the optimum number of workers to have making widgets.

6. Refer to Exercise 5. Suppose the widget manufacturing plant determined that the demand for widgets was such that a second painting facility was justified. Determine the correct number of new workers to hire to make the additional widgets.

Solutions

5.

```
      SIMULATE
WORKER FUNCTION  RN1,D7
.05,20/.17,21/.35,22/.56,23/.81,24/.96,25/1,26
PAINT FUNCTION   RN1,D5
.1,6/.25,7/.7,8/.93,9/1,10
NUMBER GENERATE  ,,,2              PROVIDE WORKERS
UPTOP ADVANCE    FN(WORKER)        ASSEMBLE A WIDGET
      ADVANCE    1,.4              TAKE WIDGET TO PAINT AREA
      QUEUE      WAIT              QUEUE FOR PAINT SHOP
      SEIZE      PAINT             USE THE PAINT SHOP
      DEPART     WAIT              LEAVE THE QUEUE
      ADVANCE    FN(PAINT)         PAINT A WIDGET
      RELEASE    PAINT             FREE THE PAINTER
      ADVANCE    1,.4              RETURN TO WORK STATION
      TRANSFER   ,UPTOP            READY TO MAKE ANOTHER WIDGET
      GENERATE   480*20            SIMULATE FOR 20 SHIFTS
      TERMINATE  1
      START      1
      CLEAR
NUMBER GENERATE  ,,,3              RUN FOR THREE WORKERS
      START      1
      CLEAR
NUMBER GENERATE  ,,,4              RUN FOR FOUR WORKERS
      START      1
      CLEAR
NUMBER GENERATE  ,,,5              RUN FOR FIVE WORKERS
      START      1
      CLEAR
NUMBER GENERATE  ,,,6              RUN FOR SIX WORKERS
      START      1
      END
```

Table 10.8 gives the results of the simulation and the cost calculation results.

Table 10.8 Results of Simulation

no workers	widgets made/day	gross profit	fixed costs	workers' salaries	net profit
2	28.75	$367	$75	$152	$140
3	42.75	$545	$75	$228	$242
4	54.35	$706	$75	$304	$327
5	59.60	$760	$75	$380	$305
6	59.6	$760	$75	$456	$229

As can be seen the optimum number of workers to have is 4. This results in a profit/day of $327.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 11
Standard Numerical Attributes (SNA's)

Every time a transaction encounters a certain block such as a QUEUE, ENTER or SEIZE block, certain statistics are gathered and kept for printing out at the end of the program. These are known as Standard Numerical Attributes (SNA's) and can be used by the programmer in other blocks when the program is being run. When the QUEUE, SEIZE, and ENTER blocks were introduced, the various SNA's associated with them were also given but not used much to this point. There are, however, many uses for them and these will become apparent as more GPSS blocks are presented. For example, the length of a queue may be used to see if a transaction will enter the queue or not; a facility is either in use or idle. This is denoted by a 1 or a 0. If a facility is being used, a transaction may be sent to a different block. As another facility is used by more and more transactions, the speed at which it operates may decrease. We shall learn how to use these SNA's to greatly increase our programming skills.

To illustrate what some of the SNA's are, consider a portion of a program having a queue, a facility and a storage as follows. The program has to do with cars arriving for minor service at a repair shop that has three service bays. The arrival rate is a car every 100 ± 23 seconds. The cars are all first inspected by a single inspector who takes 50 ± 6 seconds to inspect the cars. The minor repairs are done in only 25 ± 7 seconds.

```
      SIMULATE
      STORAGE    S(REPAIR),3     THREE SERVICE BAYS
      GENERATE   100,23 CARS     COME FOR REPAIRS
FIRST QUEUE      LINE            FORM QUEUE FOR INSPECTION
      SEIZE      BILL            BILL IS THE INSPECTOR
      DEPART     LINE            LEAVE THE QUEUE
      ADVANCE    50,6            BILL DOES THE INSPECTION
      RELEASE    BILL            FREE BILL
SECND ENTER      REPAIR          THREE SERVICE BAYS AVAILABLE
THIRD ADVANCE    25,7            NEXT SERVICE
      LEAVE      REPAIR          LEAVE STATION
```

Assume that the program is in the process of being executed so that all of the various blocks have statistics associated with them. These are the SNA's referred to above. There are also quite a few more that will be given each time a new block is introduced.

A list of the ones that are associated with the blocks we have learned so far is repeated next. The

explanation of each SNA will refer to the program given above. In case the block is referred to by a number such as QUEUE 5, a SNA reference would be given by Q5 for the SNA Q. If the block reference is a label, as it is in all of our examples so far, the SNA reference is given by using parenthesis, such as Q(LINE).

GPSS Standard Numerical Attributes

QUEUES

Name	Example	Meaning
Q(name)	Q(LINE)	Current queue content. This might be 3 cars waiting for BILL
QA(name)	QA(LINE)	The average queue content. If the average queue length of the queue LINE was 2.3 cars, this is 2.300.
QC(name)	QC(LINE)	Queue entry count; every time a QUEUE block is entered, this is incremented. For 32 cars having entered the shop, this would be 32.
QM(name)	QM(LINE)	Maximum queue content. This is the maximum number of cars that were ever in the queue.
QT(name)	QT(LINE)	Average resident time in the QUEUE LINE. This includes all cars that enter the QUEUE block, even if there is no queue and the car passes immediately through it to the next block. This statistic might be computed as follows: if 40 total cars entered the shop and the total time in the queue for all cars was 1200, this would be 1200/40 or 30.0000.
QX(name)	QX(LINE)	The average residence time in the queue. This does not include zero entries. Thus, for the 40 trucks in the previous SNA, if 10 entered the QUEUE block only to find the facility BILL free, this would be 1200/30 or 40.0000.
QZ(name)	QZ(LINE)	Zero entry count. This would be 10 for the data in the example here.

FACILITIES

Name	Example	Meaning
F(name)	F(BILL)	0 if not captured; 1 otherwise. If BILL is busy when another car comes, this is 1; else 0.
FC(name)	FC(BILL)	Number of times the facility has been captured. If 40 cars entered the service area, this is 40.
FR(name)	FR(BILL)	Facility utilization, in parts per thousand. If BILL was busy for 300 time units and 400 has passed, this would be 750.

FT(name)	FT(BILL)	The average facility holding time.

STORAGES

Name	Example	Meaning
R(name)	R(REPAR)	Remaining storage capacity. If one car is being fixed, this is 2. (3 - 1)
S(name)	S(REPAR)	Current storage content. This would be 1.
SA(name)	SA(REPAR)	Integer portion of the average storage content.
SC(name)	SC(REPAR)	Storage entry count - each time an ENTER block referencing the storage is executed, this is increased by 1. Thus, if 41 cars had entered for repairs, this is 41.
SM(name)	SM(REPAR)	Maximum storage content - this is the maximum value Sj has attained. Since there were 3 repair facilities, this cannot be larger than 3.
SR(name)	SR(REPAR)	Storage utilization, in parts per thousand. If the repair facilities were used .567 of the time, this is 567.
ST(name)	ST(REPAR)	The average holding time.

note: it is also possible to reference SNA's by using a single dollar sign, "$". Thus, Q$WAIT and Q(WAIT) have identical meanings. Since the use of parentheses is easier to understand, this practice will be followed here.

OTHER SNA'S

There are many other SNA's. Some have been encountered already without specifically referring to them as such. These are known as `system SNA's'.

W(FIRST) - is the number of transactions currently at the block with the label FIRST. If there are four cars in the QUEUE LINE block, this is 4. This is exactly the same as the SNA Q$LINE. However, most blocks do not have such a SNA and the W$name must be used.

N(SECND) - is the total number of transactions which have entered the block with the label SECND. If 54 cars have entered the service area, this is 54.

C1 - is the relative clock.

AC1 - is the absolute clock.

TG1 - is the current value of the termination counter.

RNj - Random number from 0 to 1. This was used in defining functions. This can also be written as RN(j). If this is used in connection with a function, the value returned is from the interval [0, 1) i. e., from 0.000000 to 0.999999. If used in any other context, the value returned is from 000 to 999.

Althoughugh reference to SNA's is by parenthesis, it is also possible to reference them by means of the single dollar sign, $. Thus, one could write Q$FIRST which is the same as Q(FIRST). This will not work when the entity is given a number and not a name. For example, if the queue block is QUEUE 5, then reference to the queue length is given by either Q(5), Q5 but not Q$5. This is an old method of referencing in GPSS and will not be used here.

Constants are SNA's and have their own "family name". This is the letter K in front of them, i. e., 3 is K3, 501 is K501 etc. This option is rarely used. Actually, about the only time students ever use this is to see if it really does work. It is not natural to write a number with a K in front of it, so this is considered as obsolete.

M1 whenever a transaction enters the system, it is tagged with the time of entry. Whenever M1 is then referenced, this value is subtracted from the current clock value. M1 is the difference between these two times. Suppose the time of entry was 5040 and when M1 is referenced, the clock is now at 5880. M1 will be 840 or 5880 - 5040. M1 is a floating point number.

The total number of SNA's may seem a bit staggering, especially since we have not yet learned what we can do with the SNA's. However, as we learn more GPSS statements and blocks, the use of the SNA's will become apparent.

Arithmetic Expressions in GPSS/H

SNA's can be used in operands in arithmetic expressions. The following operations are used in GPSS/H:

- `+` unary and binary addition
- `-` unary and binary subtraction
- `/` division
- `*` multiplication
- `@` modular division

The above operations are all familiar to us with the possible exception of the modular division. This is defined as division where only the remainder is kept. For example, 7@4 is 3 since the remainder is 1, 9@10 is 9 (0 with a remainder 9), etc. Thus, one could have:

```
ADVANCE N(BLOCKA)*Q(FIRST)+Q(LAST)*3.5
```

Since the arithmetic operations in GPSS/H are so similar to those encountered in other programming languages, not much more will be said of them at this time.

Several examples of their possible use are given next.

Any SNA can be used in a program as an operand. The use of SNA's will greatly expand one's ability to build meaningful simulation models. As additional blocks are introduced, it will become even more apparent how useful they are in writing simulation models. Several examples are given next. Some might appear to be quite fanciful but they illustrate the extreme power and flexibility of the language.

```
a) TRUCK GENERATE  ,,,4
          ADVANCE   60*N(TRUCK)-60
```

Four transactions will leave the GENERATE at time 0. The first is put on the FEC chain for a time of 60*N(TRUCK)-60. The block count when the first transaction has left is 1. Therefore, the time on the FEC is 0. The second transaction will be put on the FEC chain for a time of 60; the third for a time of 120 and the 4th for 180 time units. The effect of this is to delay the entry of the transactions after the first by a factor of 60 time units.

```
b) BLOCKA SEIZE     TOMMY
          .......
          .......
          ADVANCE   N(BLOCKA)*2
```

Transactions entering the ADVANCE block will be put on the FEC for a time equal to 2 times the number of transactions that have entered the SEIZE block with the label BLOCKA. The above ADVANCE block could have been written:

```
ADVANCE  FC(TOMMY)*2
```

because FC(TOMMY) give the number of times the facility TOMMY has been captured.

```
c) TERMINATE  W(BLOCKC)
```

The counter given by the START n statement is decremented by the amount equal to the current block count at the block with the label BLOCKC.

```
d) ADVANCE    2.5*AC1
```

The transaction will be put on the FEC for a time equal to 2.5 times the absolute clock value.

```
e) ADVANCE    QA(LINE)
```

The transaction will be placed on the future events chain for a time equal to the integer portion of the average queue content of the queue named LINE.

```
f) ADVANCE    Q(STORE)
```

A transaction entering the ADVANCE block will be placed on the FEC for a time equal to the length of the queue STORE. Most of the programs from now on will make use of SNA's.

Example 11.1

Customers arrive at Joe's barber shop every 15 ± 6.5 minutes. This distribution is constant throughout the day. If no customers are waiting, Joe will tend to take his time cutting hair. As customers arrive and fill up the shop, Joe will speed up his hair cutting. The time it takes Joe to cut hair is given by the following:

people in queue	time to give hair cut
0	18
1	16
2 or 3	14
4 or 5	13
more than 5	12

Simulate the operation of Joe's barber shop for 5 straight shifts of 480 minutes each.

The program to do the simulation is as follows:

```
      SIMULATE
TIME  FUNCTION   Q(WAIT),D7
0,18/1,16/2,14/3,14/4,13/5,13/6,12
      GENERATE   15,6.5          PEOPLE ARRIVE
```

```
      QUEUE      WAIT           WAIT IN SEATS
      SEIZE      JOEB           ENGAGE JOE FOR HAIR CUT
      DEPART     WAIT           LEAVE THE SEAT
      ADVANCE    FN(TIME)       CUT HAIR
      RELEASE    JOEB           FREE THE BARBER
      TERMINATE                 LEAVE THE SHOP
      GENERATE   480*5          SIMULATE FOR 5 SHIFTS
      TERMINATE  1              END OF SIMULATION
      START      1
      END
```

A portion of the output is given next:

```
RELATIVE CLOCK: 2400.0000 ABSOLUTE CLOCK: 2400.0000

BLOCK   CURRENT   TOTAL
1                 158
2       4         158
3                 154
4                 154
5       1         154
6                 153
7                 153
8                 1
9                 1

        --AVG-UTIL-DURING--

FACILITY   TOTAL    ENTRIES   AVERAGE
           TIME               TIME/XACT
JOEB       0.995     154       15.506

QUEUE     MAXIMUM   AVERAGE   TOTAL   ZERO    AVERAGE   $AVERAGE
          CONTENTS  CONTENTS  ENTRIES ENTRIES TIME/UNIT TIME/UNIT
WAIT        4        1.750     158      1      26.582    26.751
```

As can be seen, Joe is kept quite busy since his time to give a haircut is often slower than the arrival rate of his customers. The maximum content of the queue was 4. Each person had to wait for over 26 minutes and the average time to give the haircuts was 15.5 minutes.

The PUTPIC Statement

Up to now the output from a GPSS/H program was the complete report that was placed in a file name.GPS where "name" is the name of the GPSS/H program. This contains all the statistics and SNA's associated with the various blocks. This needs to be interpreted. If the output is to be included in a report it needs to be edited. Often only a few SNA'a are needed from the output and it is desired to have a additional text or symbols with the output. The SNA's needed from a program might be the utilization of a machine, the number of units produced, the number of workers in the factory, the average number of people in a queue for the day, etc. The way to print out only selected SNA's as well as customize the output is provided by the PUTPIC statement. The name comes from PUT a PICture on the screen. The general form of it is:

```
(label) PUTPIC LINES=n,FILE=file name,list of SNA's
line 1 text plus field specifications for SNA
line 2 " "
line 3 " "
......
......
line n " "
```

The label is optional and is rarely used.

The LINES=n give the number of lines of output. This is an integer which must correspond to the number of lines in the output picture, i. e., if n=8, there are exactly 8 lines in the output.

FILE = file name is where the output from the PUTPIC is to be directed. If this is omitted, the output goes directly to the screen. Since there generally is more output than can fir on a single screen, if this option is used, the output will scroll rapidly past the viewer. Thus, this option is not used too often. If the form of this is: FILE=SYSPRINT, the output is added to the normal report. If START 1,NP was used, which is common if the PUTPIC is used, the output list file will contain the normal program listing with statements and blocks numbered, and the output from the PUTPIC statement. The normal output is not included. This is the most common use of the PUTPIC statement. This output is in a form that can easily be edited using any of the common text editors such as DOS 5 EDIT, WordPerfect, MicroSoft WORD, etc.

If any other file specification is used such as FILE=FIRST.OUT, the output from the PUTPIC statement is sent to this file. This is the only output to be placed in this file.

Often the PUTPIC statement is too long to fit on a single line as it is limited to position 72. In this case, the statement can be continued by using the underscore "_" just as is done for blocks. Remember that because of this method of continuing a line in GPSS/H, the underscore cannot be

output as part of the PUTPIC statement.

The list of SNA's to be printed is the last part of the PUTPIC statement. These are a list of SNA's to be printed out separated by commas. It is all right to have these SNA's placed in parentheses. Also, the SNA's can have arithmetic associated with them. Some examples of the first line of PUTPIC statement are:

```
a)  PUTPIC  LINES=3,FILE=SYSPRINT,(QA(WAIT),FR(MACH1)/10)
b)  PUTPIC  LINES=10,FILE=FIRST.OUT,N(BLOCKA),N(BLOCKB),_
      FR(MACH1),FR(MACH2),QA(WAIT))
c)  PUTPIC  LINES=1,FR(MACH1)/10,SR(TUGS)/10
```

In a), there will be 3 lines of output. The output will be part of the normal system report. The SNA's printed out are the average time spent in the queue WAIT and the utilization of the facility MACH1 at a percent. Notice the use of parentheses and arithmetic for the SNA's.

In b), there will be 10 lines of output. This will be sent to the file FIRST.OUT. The output will include values for the SNA's N(BLOCKA), N(BLOCKB), FR(MACH1), FR(MACH2) and QA(WAIT).

In c), there will be only one line of output for the SNA's FR(MACH1) and SR(TUGS) as percents. The output will come directly to the screen.

The n-lines below the PUTPIC statement can have text (except for the underscore, as noted, and asterisks). This text is printed out exactly as written and will appear in the output report. As an example consider the following. (Assume that the fractional utilization of the facility MACH1 is 893).

```
     PUTPIC LINES=5,FR(MACH1)/10
|=================================================|
|        << RESULTS OF SIMULATION STUDY!!>>        |
|                                                 |
|     THE UTILIZATION OF THE FIRST MACHINE WAS     |
|                     ***.**%                     |
|=================================================|
```

The output will be printed on the screen as:

```
|=================================================|
|        << RESULTS OF SIMULATION STUDY!!>>        |
|                                                 |
```

```
|     THE UTILIZATION OF THE FIRST MACHINE WAS     |
|                       89.3%                      |
|==================================================|
```

It will appear on the screen as typed, i. e., if the first "|" was 10 spaces from the left, so will the first be positioned.

There is only one carriage control character associated with the PUTPIC statement and that is a zero in position 1 of the first line. The results in a blank line being inserted before the first line of the output from the PUTPIC statement.

The list of SNA's will be placed wherever there are asterisks in the same order as they are written on the PUTPIC line. Thus,

```
 PUTPIC  LINES=7,FILE=SYSPRINT,(N(BLOCKA),N(BLOCKB),_
      FR(WORKER)/10)

 ######################################################
 #             RESULTS OF SIMULATION                  #
 #  NUMBER OF ENTRIES INTO MACHINE A     ***          #
 #  NUMBER OF ENTRIES INTO MACHINE B     ***          #
 #  UTILIZATION OF THE WORKER            ***.**%      #
 ######################################################
```

would result in output such as: (assuming values for the SNA's as given):

```
 ######################################################
 #             RESULTS OF SIMULATION                  #
 #  NUMBER OF ENTRIES INTO MACHINE A     325          #
 #  NUMBER OF ENTRIES INTO MACHINE B     645          #
 #  UTILIZATION OF THE WORKER             88.76%      #
 ######################################################
```

If the field specification is smaller than the actual value of the SNA, the value is still printed out but the right margin is distorted. For example, if the value of N(BLOCKb) in the previous example was 1234 instead of 645, the output would be:

```
 ######################################################
 #             RESULTS OF SIMULATION                  #
 #  NUMBER OF ENTRIES INTO MACHINE A     325          #
```

```
#  NUMBER OF ENTRIES INTO MACHINE B    1234       #
#  UTILIZATION OF THE WORKER             88.76%   #
##################################################
```

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 12
The TEST block

Up to this point transactions moved through the various systems sequentially from block to block. The only way we had to route them to different blocks was via the TRANSFER block. There are many times in a model when the programmer will want to route a transaction to one block or another depending on some aspect of the system. There will also be times when the programmer will want to keep transactions from moving forward until a specific condition is met. Both of these are done using the TEST block.

It is possible to do a test on two SNA's and then route the transaction to one or another of two blocks depending on the result of the test. Examples of where a TEST block might be used arise frequently during a simulation. Some possible examples where a TEST block might be used are:

1. If the queue at a shop is greater than 5, arriving customers do not enter.
2. After 12 hours of working a machine is shut down for maintenance for 1/2 hour.
3. At 5 o'clock the barber locks the door on his shop, but customers already in are still served.

GPSS/H does this and can also perform a test on two SNA's and hold the transaction at the block doing the test until the test is true. Examples of these might be:

1. Ships cannot enter the harbor until one of two tug boats is free to guide it into the berth.
2. A part will not moved from machine unless the next machine has been used less than 75% of the time.
3. If it is between noon and 12:30 p.m., no customer can enter a repair facility.
4. Once a machine has finished making 500 parts it is taken out of service for repairs and maintenance. This down time lasts for two hours. Parts arriving have to wait until the repairs and maintenance are finished.

We shall see that the use of TEST blocks will greatly expand our programming ability. There are two basic forms of the TEST block.

The TEST Block in Refusal Mode

This form of the TEST block is as follows:

```
TEST R A,B
```

where R is a conditional operator that is one of the following:

symbol	meaning
L	less than
LE	less than or equal
E	equal
NE	not equal
G	greater than
GE	greater than or equal

The conditional operator must be placed one space after the word TEST. A and B are any SNA's to be tested via the conditional operator. Some examples of the TEST block might be:

```
a) TEST E Q(TOM),Q(BILL)
b) TEST NE R(DOCK),4
c) TEST L FR(MACH),400
d) TEST G W(BACK1),1
e) TEST E N(BLOCKA),N(BLOCKB)
```

The way the TEST block works when a transaction enters it is as follows. The first SNA is compared with the second using the conditional operator. If the test is true, the transaction moves to the next sequential block. If the test is false, the transaction must wait in the TEST block until some future time when the test becomes true. In addition, the transaction waits on the CEC. Thus in a), the test is "is the length of the queue named TOM equal to the length of the queue named BILL?" If the answer is yes, the transaction will move to the next block but if the answer is no, the transaction will remain in the block until such time that the test is true.

In b), the remaining storage of DOCK must be not equal to 4 before the transaction can move to the next block. Similarly, for c), the fractional utilization of the facility MACH must be less than .400 or the transaction will reside in the TEST block until it is.

In example d), the transaction will test to see if the current count of the block BACK1 is great than 1. Unless it is greater than 1, the transaction will not leave the test block.

In e), the transaction will be held until the total block count of the block labeled BLOCKA is equal to the count of the block labeled BLOCKB.

The next example should be studied to understand how a TEST block in refusal mode can be

used.

Example 12.1

Joe cuts hair in 15 ± 6 minutes. Customers arrive every 14 ± 8 minutes. Joe has only one chair for them to wait so if a customer arrives to find this taken, he will leave. Of the people who leave, 30% will wait for 30 ± 12 minutes and return to see if a chair is free. (If the chair is not available this second time, he will again leave and 30% of the time will wait for 30 ± 12 minutes and then return, etc.). The rest will go away. Joe works from 8 to 5 with no time off for lunch. At 5 Joe locks the door but will finish cutting the hair of anyone in his shop. Simulate for a typical day.

The program to do the simulation is given below:

```
        SIMULATE
        STORAGE    S(SEATS),1
        GENERATE   14,8                 CUSTOMERS ARRIVE
        TEST L     N(TIME),1            IS IT PAST 5 YET?
BACK    TRANSFER   BOTH,,AWAY           IS THERE ROOM IN THE SHOP?
INSHOP  ENTER      SEATS                TAKE A SEAT
        SEIZE      JOEB                 ENGAGE JOE
        LEAVE      SEATS                LEAVE THE SEATS
        ADVANCE    15,6                 RECEIVE HAIRCUT
NDONE   RELEASE    JOEB                 FREE JOE
        TERMINATE  LEAVE                THE SHOP
AWAY    TRANSFER   .7,,GONE             70% LEAVE
        ADVANCE    30,12                30% WAIT
        TRANSFER   ,BACK                RETURN TO SHOP
GONE    TERMINATE  LEAVE                SYSTEM
TIME    GENERATE   480                  5 O'CLOCK COMES
        TEST E     N(NDONE),N(INSHOP)   IS SHOP EMPTY
        TERMINATE  1                    YES, JOE CAN GO HOME
        START      1
        END
```

There are several TEST blocks in the program. The first is used to stop any customer transactions from entering the shop after 5 o'clock. The simulation starts at time 0 and, with one minute as the basic time unit, 5 o'clock will be given by time 480. At this time no more customers will be allowed in the shop. This block is:

```
      TEST L     N(TIME),1
```

At simulated time 480 a transaction leaves the block

```
      TIME GENERATE 480
```

This make its block count 1. The TEST block is then false and any transactions that enter it will be held up and not be allowed to proceed. This corresponds to not allowing any customers to enter the shop after 5 o'clock.

note: an alternate block to use would have been

```
      TEST L  AC1,480
```

The second TEST block is the timer transaction segment of the program. At time 480 the timer transaction arrives to shut off the program. This corresponds to it being 5 o'clock. However, the transaction first enters the TEST block

```
      TEST E N(INSHOP),N(NDONE)
```

INSHOP is the label for the ENTER block which corresponds to a customer entering the shop. N(NDONE) corresponds to the total number of customers who have left the shop. It is the label for the RELEASE JOEB block. Suppose that at simulated time 480, N(INSHOP) was 33 and N(NDONE) was 31. The effect of the second TEST block would then be to hold the timer transaction until all of the customers in the shop had been serviced by Joe.

The program was run and selected portions of the output are:

```
RELATIVE CLOCK: 494.9533 ABSOLUTE CLOCK: 494.9533

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1      1        32     11              1
2               31     12              1
BACK            32     GONE            1
INSHOP          30     TIME            1
5               30     15              1
6               30     16              1
7               30
NDONE           30
9               30
```

```
AWAY            2

          --AVG-UTIL-DURING--

FACILITY    TOTAL    ENTRIES    AVERAGE
            TIME               TIME/XACT
JOEB        0.868      30        14.325

          --AVG-UTIL-DURING--

STORAGE     TOTAL    ENTRIES    AVERAGE    CAPACITY    AVERAGE
            TIME               TIME/UNIT              CONTENTS
SEATS       0.222      30        3.669        1         0.222
```

Joe was busy for 86.8% of the time. A total of 30 customers entered the shop and received haircuts. Joe worked until 14.9533 minutes past five to finish cutting the hair of the customers in his shop.

Whenever a TEST block in refusal mode is used in a program, great care must be exercised that the transaction does not remain in the block forever, if this is not the programmer's desire. There is another caution in using this block that we have not been too concerned with up to this time. Whenever a transaction is in a blocked condition at a TEST block, it remains on the current events chain. Whenever the processor does a re-scan this block must be tested. This can be quite costly in terms of execution time. In some cases there will be ways to avoid using such inefficient blocks. The TEST block is both convenient and easy to understand. However, if it is possible to avoid using it, alternate programming should be used. Some other blocks that might be used in its place will be introduced later. In some cases, there is no other method available other than the TEST block.

TEST Block in Normal Mode

The other form of the TEST block has a C operand. The form of it is simply:

```
TEST R A,B,C
```

where C is the name of a block the transaction is routed to if the test is false. Thus,

```
TEST E Q(TOMMY),Q(SALLY),DOWN
```

will test the queue length of the queue TOMMY the queue length at SALLY. If they are equal the transaction will go to the next sequential block. If they are unequal, the transaction will go to the

block named DOWN. For many programmers, who are used to the logic of Fortran, the way GPSS works for the TEST block is going to seem to be quite the opposite to what one would expect. Thus, great care is required when using this block.

Example 12.1

In a manufacturing process, parts come to a machine for forming. The interarrival rate is 14 ± 7.5 minutes. There are two machines available for forming. The first can form in 16 ± 5.4 minutes and the other takes considerably longer as it takes 24 ± 8 minutes to form. In fact, this second machine is in such poor condition that it is not used until the first machine is utilized to its fullest. This faster machine cannot be used more that 85% of the time or it may overheat. Parts enter the room where both machines are located and use the faster machine until it reaches the 85% utilization at which time the slower machine is used until the utilization is again below 85%. Build a GPSS/H model to represent the system. Simulate for 20 straight shifts of 8 hours (480 minutes).

Solution

The GPSS/H program to do the simulation is:

```
      SIMULATE
      GENERATE    14,7.5
      QUEUE       WAIT
      TEST LE     FR(MACH1),850,DOWN1
      SEIZE       MACH1
      DEPART      WAIT
      ADVANCE     16,5.4
      RELEASE     MACH1
      TERMINATE
DOWN1 SEIZE       MACH2
      DEPART      WAIT
      ADVANCE     24,8
      RELEASE     MACH2
      TERMINATE
      GENERATE    480*20
      TERMINATE   1
      START       1
      END
```

Edited portions of the output are:

```
RELATIVE CLOCK: 9600.0000 ABSOLUTE CLOCK: 9600.0000

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               685    11     1        177
2               685    12              176
3      1        685    13              176
4               507    14              1
5               507    15              1
6      1        507
7               506
8               506
DOWN1           177
10              177

          --AVG-UTIL-DURING--

FACILITY  TOTAL   ENTRIES   AVERAGE
          TIME              TIME/XACT
MACH1     0.850     507      16.102
MACH2     0.440     177      23.854

QUEUE    MAXIMUM   AVERAGE   TOTAL   ZERO    AVERAGE   $AVERAGE
         CONTENTS  CONTENTS  ENTRIES ENTRIES TIME/UNIT TIME/UNIT
WAIT        4        0.616     685     270      8.627     14.239
```

Notice that machine one was busy for the maximum allowable time, namely, 85%. Machine two was only busy 44% of the time. The maximum queue was 4 but the average queue length was only 0.616.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 13
GPSS/H built in functions

In Chapter 10 functions were introduced. By using piece-wise linear approximations it is possible to approximate any continuous functions. Two distributions that are very commonly used in simulations are the Poisson (exponential) and normal (Gaussian) distribution. These arise in the simulation studies of a great many systems. For example, the interarrival rates of telephone calls is often given by the exponential distribution, the times to travel from point A to point B by truck is normally distributed, the time for a ship to return to a port is exponential, the time between storms is exponential, etc. It is assumed that both of these functions are well known to students of simulation.

Since these functions are so commonly referred to in simulation studies, they are built into GPSS/H and sampling from them is quite easy.

Poisson Distribution

The Poisson distribution is a one parameter distribution being completely specified by its mean value. The built in function to be used in sampling from it is given by:

```
RVEXPO(random no. stream, mean)
```

where the random no. stream is the number of the random number stream to be used in obtaining the sample. Recall that GPSS/H has nearly an infinite number of these but normally one only uses small numbers such as 1, 2, 3, etc.

Examples of this are:

a) `ADVANCE  RVEXPO(1,12.3)`
b) `GENERATE  RVEXPO(1,3.4)`

In a), the transaction is placed on the FEC for a time given by sampling from the exponential distribution with mean 12.3.

In b), transactions are generated at times given by sampling from the exponential distribution with mean 3.4.

Normal Distribution

The normal distribution is a two parameter distribution and is specified by the mean and standard deviation. The GPSS/H built in function to sample from the normal distribution is given by:

```
RVNORM(random no. stream, mean, std. dev.)
```

where random no. stream refers to the number of the random number stream to sample from (as in RVEXPO).

Examples of this might be:

```
a) ADVANCE     RVNORM(1,20,2.3)
b) GENERATE    RVNORM(1,30,5.5)
```

In a), the transaction is placed on the FEC for a time that is obtained by sampling from a normal distribution with mean of 20 and std. dev. of 2.3. In b), transactions are generated according to the normal distribution with mean of 30 and std. dev. of 5.5.

GPSS/H samples from a distribution bounded by 44 standard deviations above and below the mean, so, while it is theoretically possible to obtain samples that are negative this is rare.

The Triangular Distribution

GPSS/H has another built in function which represents the triangular distribution. This distribution is one that look like a triangle with one side on the x-axis. The side extends from a minimum value to a maximum value. The most likely value is the mode. A triangular distribution with a minimum of 10, a maximum of 100 and a mode of 20 will be skewed to the left. A triangular distribution with minimum value of 10, maximum of 100 and a mode of 80 will be skewed to the right. If the mode of this distribution was 55, the distribution would be symmetrical. To sample from a triangular distribution in GPSS one uses the built in function:

```
RVTRI(random no. stream, min., mode, max.)
```

Thus,

```
ADVANCE RVTRI(2,0,3,10)
```

will sample from the triangular distributionn having minimum value of 0, mode of 3 and maximum

value of 10.

Using Exponential and Normal Distributions Other than Built in Ones

As indicated, there may be times when the built in functions cannot be used. In that case, you have to supply the exponential and normal distributions as piece wise continuous functions.

The exponential distribution can be put in a form so that giving a random sample in the interval [0,1), a corresponding interarrival time is given by:

$$T_{sample} = T_{mean} (\ln 1 /(1 - RN))$$

where T_{mean} = mean value of distribution
RN = random number

The 24 piece wise approximation for the distribution

$$\ln 1/(1 - RN)$$

to be used in sampling from the exponential distribution is as follows:

```
EPDIS FUNCTION   RN1,C24
0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38
.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9998,8
```

Normally one samples from the exponential distribution using either a GENERATE or an ADVANCE block. Some examples of how this can be done is as follows:

```
a) GENERATE   100,FN(EPDIS)
b) ADVANCE    225,FN(EPDIS)
```

In a), a transaction is generated by sampling from the exponential distribution with a mean of 100.

In b), a transaction is placed on the FEC for a time specified by sampling from the exponential distribution with a mean of 225.

Notice that when either a GENERATE or ADVANCE block has a function as the B operand, the effect if to multiply the value in the A operand by the value of the function. This is true whenever a function is in the B operand, not only for the exponential distribution. Thus, if you had

```
ADVANCE    FN(ONE),FN(TWO)
```

and the value of FN(ONE) is 20 and FN(TWO) is 8, the transaction is put on the FEC for 160 time units, not for a time obtained by sampling from the distribution 20 ± 8.

The normal distribution is a two parameter distribution. These parameters are the mean and standard deviation. A standard normal distribution is one whose mean is 0 and standard deviation is 1. To sample from such a distribution one first samples from the standard normal distribution and then converts the result to the non-standard distribution. This is done as follows:

Sample value = (std. dev.) (value from SNP) + mean

SNP is the value drawn from the standard normal population.

The piece wise continuous function used in sampling from the SNP is:

```
SNORM FUNCTION   RN1,C25
0,-5/.00003,-4/.00135,-3/.00621,-2.5/.02275,-2
.06681,-1.5/.11507,-1.21/.15866,-1/.21186,-.8/.27425,-.6
.34458,-.4/.42074,-.2/.5,0/.57926,.2/.65542,.4
.72575,.6/.78814,.8/.84134,1/.88493,1.2/.93319,1.5
.97725,2/.99379,2.5/.99865,3/.99997,4/1,5
```

Examples of this are:

```
a) ADVANCE     3.2*FN(SNORM)+20.7
b) ADVANCE     10*FN(SNORM)+100
c) GENERATE    2.2*FN(SNORM)+20.8
```

In a), the transaction is placed on the FEC for a time given by sampling from the normal distribution with a mean of 20.7 and a std. dev. of 3.2. In b), the transaction is placed on the FEC for a time given by sampling from the normal distribution with a mean of 100 and a standard deviation of 10. In c), transactions are generated at times obtained from sampling from the norma distribution with mean of 20.8 and std. dev. of 2.2.

Notice that the function SNORM, used in sampling from the normal distribution can return a value as low as -5. Suppose one had

```
ADVANCE     2.1*FN(SNORM)+10
```

and just such value was returned. The resulting time is -2.1*5 + 10 or -.5 time units. This is meaningless as one cannot go back in time and an execution error would result. It is necessary to guard against this by always insuring that the standard deviation is less than 1/5 (20%) of the mean. Alternately, one can add other GPSS code to test for negative times and filter them out. (These tests will be discussed later).

Exercises

1. A careful study of the widget and squidget manufacturing plant you so carefully designed previously has revealed that your actual times to build a squidget are not uniformly distributed. In addition, the time to fire them is also not uniformly distributed. The actual distributions are as follows:

assembly time (min)	rel freq
25	.01
26	.03
27	.05
28	.10
29	.18
30	.26
31	.18
32	.10
33	.05
34	.03
35	.01

Oven times:

oven use time (min)	rel freq
6	.05

7	.25
8	.40
9	.25
10	.05

Re-do the exercise with the above data.

2. The data for making a squidget has been changed. Further careful study by you indicates that it is normally distributed with a mean of 30 std dev. of 20% of this. The time to fire the oven is also normally distributed with a mean of 8, std dev. of 20%. Do you answers change much?

3. Change the data in Exercise 2. to be exponential with mean of 30 for assembling and 8 for firing. Now how do the answers change? (remember that you are to run the program for 4, 5, 6, etc. assemblers).

4. A manufacturing system consists of two waiting lines and two servers. Only 4 units can wait at station 1 and two at station 2. If another unit comes along when the waiting space at station 1 is full, it leaves and a penalty in incurred.

```
>========> O O O O ====> station 1 ====> O O station 2 ====>
    |
    |
   =====> units turned away
```

Arrivals are exponentially distributed with a mean of 0.4 time units. Service times are exponentially distributed at both stations with means of 0.25 and 0.5 respectively. Model this system to see how efficient it is. Simulate for 500 time units.

5. Change the station working times in Exercise 4 to have means of .35 for station 1 and .40 for station 2. Note that their sum is still 0.75 as it was before. Is the system improved?

6. Repeat Exercise 4 but now with waiting space for the stations allocated as 3 and 3.

7. Repeat Exercise 6 but now with the times changed as in Exercise 2.

8. Repeat Exercises 4 to 7 but with the interarrival and service time Erlang, order 2.

Solutions

```
5.    SIMULATE
      STORAGE     S(ONE),4/S(TWO),2      4 AND 2 STORAGES
      GENERATE    RVEXPO(1,.4)           UNITS ARRIVE
      TRANSFER    BOTH,,AWAY             IS THEIR ROOM
      ENTER       ONE                    YES, JOIN FIRST QUEUE
      SEIZE       MACH1                  DO SERVICE 1
      LEAVE       ONE                    LEAVE THE QUEUE
      ADVANCE     RVEXPO(1,.35)          PERFORM SERVICE
      ENTER       TWO                    IS THERE A SEAT FREE?
      RELEASE     MACH1                  YES, FREE MACHINE
      SEIZE       MACH2                  USE MACHINE 2
      LEAVE       TWO                    LEAVE SEAT 2
      ADVANCE     RVEXPO(1,.4)           PERFORM SERVICE
      RELEASE     MACH2                  FREE THE SECOND SERVER
      TERMINATE   AWAY                   IT GOES
AWAY  TERMINATE                          SORRY, SYSTEM FULL
      GENERATE    500                    500 TIME UNITS
      TERMINATE   1                      END OF PROGRAM
      START       1
      END
```

8. Most queueing theory problems have no exact solutions. A few, however, do and these can be studied to compare the simulation solution with the expected result obtained by using a formula. One such example is the exercise presented next.

A one car wash facility has cars arrive with an average interarrival time of 5 minutes (use 300 seconds). The washing time is 4 minutes (240 seconds). If a car arrives and there is no waiting space, it will leave and not return (or if it does return, it is considered as a "new" car). Determine the behavior of the system for one, two and three waiting spaces.

The exact solution for this problem gives as the fractions served:

$$\text{fraction served} = 1 - ((1 - x)/(1 - x^{m+1}))* x^{m}$$

where x is the utilization factor which is the ratio of the mean service time to mean interarrial time. M is the number of waiting spaces. Thus, for an single waiting space and X = 4/5 (as given above), the above formula gives: 0.738 as the theoretical number served. Determine how many simulations are needed to approach this number.

9. A particular worker tends to work at a slower rate as the 8 hour day goes by. During the first 2

hours, it takes him 12 minutes to perform a service; during the next 2 hours, his average service time is 15 minutes; during the fifth, sixth and seventh hours, each service takes him an average of 17 minutes, Service started during the eighth hour requires an average of 20 minutes. With a mean time unit of .1 minute, define a discrete function to model this blokes service time.

Change the above to a continuous function assuming the following: at time zero, the service time is 12, by the end of the second hour, it is 15; at the end of the 4th hours, it is 17; by the end of the 7th hour ,it is 20 and finally, at the end of the eighth and last hour it is 21.

10. A job shop has 5 different machine groups. These machines are used to manufacture three different products. The number of machines available for each group are:

group	no. machines
1	**3**
2	**2**
3	**4**
4	**3**
5	**1**

The jobs come along according to the distribution:

interarrival time	relative freq	cum freq
< or = 0.0	0.0	0.0
> 0.0 < or = 0.2	0.40	0.40
> 0.2 < or = 0.4	0.30	0.70
> 0.4 < or = 0.6	0.15	0.85
> 0.6 < or = 0.8	0.10	0.95
> 0.8 < or = 1.0	0.05	1.00
> or = 1.00	0.0	

When a job comes along, 30% are type 1; 50% are type 2 and 20% are type 3. Each job type requires different sequences of machines to be used. For example, a job type 1 requires the sequence: 3, 1, 2, 5; a job of type 2 requires the sequence: 4, 1, 3 and type 3: 2, 5, 1, 4, and 3. The time for each job type on each machine are given in the table:

job type	no machines	sequence	time (exp dist)
1	4	3	0.5
1	4	1	0.6
1	4	2	0.85
1	4	5	0.50
2	3	4	1.10
2	3	1	0.80
2	3	3	0.75
3	5	2	1.20
3	5	5	0.20
3	5	1	0.70
3	5	4	0.90
3	5	3	1.00

The above times are all in hours. At the start of the simulation you have 8 jobs in the system. Simulate for 50 continuous shifts of 8 hours each.

a) simulate with the data as given above.
b) assume that the distributions are really Erlang, order 2. How does this change the results?

note: the program to do the simulation is relatively easy to write but a bit lengthy. Later, you will be shown how to do the same problem using only about 1/6th the lines of code.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul

Chapter 14
Parameters

As each transaction travels from block to block it caries with it several things. For example, we already know that transactions can have different levels of priority. In addition, there are other ways to make each transaction different. The way to do this is given next.

Each transaction posses a set of abstract things known as parameters. These are carried with the transaction as it moves through the simulation and can be modified during the program. The values of these are not normally a part of the output report but can be used during the program by the programmer. Just as transactions can be viewed conceptually as "stick people", it is possible to think of parameters as pockets on the pants of the stick people. You can put numbers inside each of the pockets to differentiate between the transactions. Each pocket has a number from 1 to 100. You can give pocket number 12 the value 4, pocket number 7 the value -234, etc. How you do this will be explained below.

Each transaction can have 4 different kinds of parameters. There can be up to 100 of each of these, although it is rare that one would use more than a few in a typical program. Parameters can be thought of as a collection of SNA's which the transaction owns. These parameters are normally numbers, although one can also give them names. For most of our purposes, they will be used only as numbers. The different types of parameters in GPSS/H are:

1. Half word parameters. This can be a number that ranges from -32,768 to +32,767. These must be integers.
2. Full word parameter. This can range from -2^{31} to $+2^{31}$ -1. The values of these is -2,147,483,648 and +2,147,483,647 and are also integers.
3. Bit word parameter. These can range from only -128 to + 127. (-2^7 to $+ 2^7 - 1$). These, too, are integers.
4. Finally, there is a floating point (decimal) parameter. The size of these are machine dependent but can be as large (or small) as $\pm 10^{35}$.

Initially, every transaction is assigned 12 half word parameters by default. Thus, although we didn't know it, all of our transactions so far had 12 of these half word parameters. The number of parameters can be increased or decreased in the GENERATE Block in position F through I. Half word parameters are indicated by nPH, full word parameters by mPF, bit word parameters by iPB and floating point parameters by jPL. It makes no difference where in positions F through I you indicate the number of each type of parameter.

Some examples of these are:

```
a) GENERATE    12,2,,,,4PH
b) GENERATE    ,,,5,,12PF,20PH
c) GENERATE    ,,,12,,5PF
d) GENERATE    12,4,,,,0PH,1PF
e) GENERATE    100,3,,,,3PH,4PF,5PB,6PL
f) GENERATE    ,,,10,1,12PL
g) GENERATE    100,,,,,,20PH,50PB
```

a) generates transactions with 4 half word parameters.

b) generates transactions with 12 full word parameters and 20 half word parameters.

c) generates transactions with 5 full work parameters and, by default, still 12 half word parameters

d) generates transactions with no half word parameters and 1 full word parameter.

e) generates transactions with 3 half word parameter, 4 full word parameters, 5 bit word transactions and 6 floating point transactions.

f) generates transactions with 12 floating point transactions.

g) generates transactions with 20 half word transactions and 50 bit word transactions.

It is important to remember that once you specify parameter types and numbers via the H - K operands of the GENERATE block, you no longer have the 12 half word parameters by default. Thus,

```
GENERATE    ,,,1,,1PL
```

generates a single transaction with only 1 floating point parameter and no half word parameters. Due to storage constraints it is best to use half word parameters unless the numbers used as parameter values are beyond the ranges. In addition, although it may not seem obvious, it is preferable to have all the transactions in a program have the same number and type of parameters. For example, you may have the main GENERATE block as GENERATE ,,,5,,20PH. Later in the program the timer transaction enters via GENERATE 480*100. It is preferable to have this as GENERATE 480*100,,,,,20PH, even though it is going to be terminated immediately.

The ASSIGN Block

Initially **the values of all Parameters are Zero. The value of a transaction's parameter can be modified via the ASSIGN block as follows:**

```
ASSIGN (parameter number),SNA,parameter type
```

where parameter number is the number of the parameter such as 1, 6, 8, etc. This can be a variable.

SNA is the value the parameter is to be given.

parameter type is either PH or PF for half word parameter or full word parameter. This can be omitted for certain cases but it is not considered good programming to do so. For example, if the transaction is given only the 12 half word parameters by default, it would be acceptable to omit the parameter type.

Thus, when a transaction leaves the block

```
ASSIGN 1,5,PH
```

parameter 1 will have the value of 5.

```
GENERATE ,,,3
ASSIGN 2,100,PH
```

Three transactions are generated and the value of their second parameter is set to 100.

```
VALUE FUNCTION   RN1,D3
.2,4/.5,7/1,8
      GENERATE   ,,,1
      ASSIGN     2,FN(VALUE),PH
```

20% of the time the transaction's second parameter will have the value of 4; 30% the value of 7 and the rest the value of 8. Since the operands of the ASSIGN block are SNA's, it is possible to have the following:

```
      ASSIGN     PH1,3,PH
```

Now what happens depend on the value already in the transaction's parameter 1. If it is 4, then parameter 4 will have the value 3.

```
TIMES GENERATE   ,,,5
      ASSIGN 1,N(TIMES),PH
```

5 transactions are created. The first will have a 1 in parameter 1, the second a 2, the third a 3, etc. This is a method of generating a number of transactions with a single GENERATE block and having each with sequential numbers in parameter 1.

```
      ASSIGN Q(WAIT)+1,1.23,PL
```

will assign the value of 1.23 to the floating point parameter given by the Q(WAIT)+1. As mentioned, it is also possible to have the following:

```
      ASSIGN TOM,10,PH
```

Later, when reference to the parameter named TOM is made, it is done as follows:

```
      ADVANCE PH(TOM)
```

The transaction will be put on the FEC for a time of 10 since the value of the parameter TOM is 10. The preference here is to use parameters given by numbers rather than by name. Since parameters are SNA's, they can be used as operands. For example, consider the following lines of code.

```
a) ADVANCE    PF4
b) TEST E     PH1,PH4,DOWN1
c) QUEUE      PH1
d) ENTER      TUGS,PB2
```

In a), the transaction will be placed on the FEC for a time given by the transaction's full word parameter number 4.

In b), a test is made to see if the first and fourth half word parameters are equal. If so, the transaction moves sequentially to the next block. It they are not equal, the transaction is routed to the block with the label DOWN1

In c), the transaction joins the queue given by its first half word parameter.

In d), the transaction will enter the storage TUGS and use a storage equal as specified by its first bit word parameter.

In addition, consider the following:

```
TIMES FUNCTION   PH1,D4
1,100/2,125/3,150/4,175
```

Now when a transaction enter the block:

```
ADVANCE    FN(TIMES)
```

it will be placed on the FEC for a time of either 100, 125, 150 or 175 time units depending on the value of its first half word parameter.

The ASSIGN block in Increment/Decrement Mode

You can add to (or subtract from) the value of a parameter by putting a plus (or minus) before the first comma in the operands:

```
ASSIGN    4+,5,PH
```

This will take the value in parameter 4 and add 5 to it.

```
ASSIGN    3-,6,PH
```

This will subtract 6 from the value in parameter 3.

```
ASSIGN    1+,Q(WAIT),PH
```

This will add the length of the queue WAIT to the transaction's first parameter. If the queue length was 4 and the value of PH1 was 12, its new value is now 16.

```
ASSIGN    7+,QX(WAIT1),PL
```

This will add the average waiting time of the non-zero entry transactions for the queue WAIT1 to the transaction's 7th floating point parameter.

Example 14.1

In Chapter 5 we had an example of a hardware store with shoppers selecting items from each of 4 possible aisles. Each shopper who went through the store took 45 ± 12 seconds to checkout. Shoppers who went directly to the checkout counter took 20 ± 8 seconds. In actual practice the time to checkout is a function of how many items each person has in the shopping cart. Suppose the

number of items selected by each person who goes down each aisle is given by the following:

aisle	no. items selected
1	3 ± 2
2	4 ± 3
3	3 ± 1
4	5 ± 4

At the checkout counter, all people select additional items as follows:

no	prob
0	.30
1	.25
2	.45

Checkout time is 3.5 seconds per item. Modify the program written in Chapter 5 to include these changes.

Solution

The program listing is given below:

```
      SIMULATE
AISLE1 FUNCTION  RN1,C2          SHOPPING IN AISLE 1
0,1/1,6
AISLE2 FUNCTION  RN1,C2          SHOPPING IN AISLE 2
0,1/1,8
AISLE3 FUNCTION  RN1,C2          SHOPPING IN AISLE 3
0,2/1,5
AISLE4 FUNCTION  RN1,C2          SHOPPING IN AISLE 4
0,1/1,10
WAIT   FUNCTION   RN1,D3
.3,0/.55,1/1,2
       STORAGE    S(CARTS),1000    PROVIDE 1000 CARTS
       GENERATE   RVEXPO(1,82.5)   CUSTOMERS ARRIVE
       TRANSFER   .12,,COUNTER     12% GO TO COUNTER
```

```
      ENTER      CARTS             REST TAKE A CART
      TRANSFER   .2,,AISLE2        80% GO TO AISLE 1
      ASSIGN     1,FN(AISLE1),PH   SELECT ITEMS IN AISLE 1
      ADVANCE    125,70            SHOP IN AISLE 1
AISLE2 TRANSFER  .25,,AISLE3       75% GO TO AISLE 2
      ASSIGN     1+,FN(AISLE2),PH SELECT ITEMS IN AISLE 2
      ADVANCE    140,40            SHOP IN AISLE 2
AISLE3 TRANSFER  .15,,AISLE4       85% GO TO AISLE 3
      ASSIGN     1+,FN(AISLE3),PH SELECT ITEMS IN AISLE 3
      ADVANCE    150,65            SHOP IN AISLE 3
AISLE4 TRANSFER  .10,,CHECK        90% GO TO AISLE 3
      ASSIGN     1+,FN(AISLE4),PH SELECT ITEMS IN AISLE 4
      ADVANCE    175,70            SHOP IN AISLE 3
CHECK ASSIGN     1+,FN(WAIT),PH    SELECT ITEMS AT COUNTER
      QUEUE      LINE              STAND IN LINE
      SEIZE      WORKER            READY TO CHECK OUT
      DEPART     LINE              LEAVE THE QUEUE
      ADVANCE    PH1*3.5           CHECK OUT
      RELEASE    WORKER            FREE THE CHECK OUT GIRL
      LEAVE      CARTS             GET RID OF CART
      TERMINATE  LEAVE             THE STORE
COUNTER ASSIGN   1,FN(WAIT),PH     SELECT ITEMS AT COUNTER
      QUEUE      LINE              STAND IN LINE
      SEIZE      WORKER            READY TO CHECK OUT
      DEPART     LINE              LEAVE THE QUEUE
      ADVANCE    PH1*3.5           CHECK OUT
      RELEASE    WORKER            FREE THE CHECK OUT GIRL
      TERMINATE  LEAVE             THE STORE
      GENERATE   3600*8*20         SIMULATE FOR 20 DAYS
      TERMINATE  1                 TIMER TRANSACTION
      START      1
END
```

Selected portions of the output are as follows:

```
RELATIVE CLOCK: 5.7600E+05 ABSOLUTE CLOCK: 5.7600E+05

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1               7026   11              5307   21              6200
```

```
2              7026   12              5307   22              6200
3              6206   AISLE4          6201   23              6200
4              6206   14              5626   COUNTER          820
5              4997   15              5626   25               820
6              4997   CHECK           6201   26               820
AISLE2         6206   17              6201   27               820
8              4631   18              6201   28               820
9      5       4631   19              6201   29               820
AISLE3         6201   20      1       6201   30               820

           --AVG-UTIL-DURING--

FACILITY   TOTAL     ENTRIES   AVERAGE
           TIME                TIME/XACT
WORKER     0.538      7021      44.114

           --AVG-UTIL-DURING--

STORAGE    ENTRIES   AVERAGE    AVERAGE   CURRENT   MAXIMUM
                     TIME/UNIT  CONTENTS  CONTENTS  CONTENTS
CARTS       6206      572.173    6.165       6         17

QUEUE      MAXIMUM    AVERAGE    TOTAL     ZERO
           CONTENTS   CONTENTS   ENTRIES   ENTRIES
LINE          9        0.371      7021      3217
```

The results of the simulation indicate that 7026 people entered the hardware store. Of these, 6206 went shopping down the various aisles. The remaining 820 went directly to the checkout counter. The checkout girl was busy 53.8% of the time. Notice that, even though this is not much at one time, there was a queue length of 9. The maximum number of carts in the shop was 17.

General Form of ASSIGN Block

There is a more general form of the ASSIGN block that is not used much any more since the exponential distribution is built in. However, it will be presented here for sake of being complete. This is:

```
     ASSIGN    A,B,C,D
```

Operands A and B have their usual meaning. C, however, is the name or number of a function. D is the type of parameter A is. If C is omitted, then D takes its place and we have the ASSIGN block presented previously. If one used all 4 operands, the effect is as follows:

1) The function as specified by the C operand is evaluated. If it returns a decimal, the value is truncated.
2) This value is then multiplied by the number in the B operand.
3) The result of the multiplication in 2) is placed in the transaction's parameter as specified by the A operand.

For example,

```
ASSIGN    3,6,5,PH
```

The function defined with the label 5 is evaluated. Suppose the result is 2. This is multiplied by 6 and the result, namely 12 is placed in the transactions 3rd halfword parameter.

```
ASSIGN    1,3,FIRST,PF
```

The function FIRST is evaluated. Suppose the number returned is 2.9543. This is truncated to 2 and multiplied by 3. The result, 6, is placed in the transaction's 1st full word parameter.

Notice that the form is as given above. If you had put:

```
ASSIGN    1,3,FN(FIRST),PF
```

a run time error would result.

The LOOP Block

GPSS/H does not have general DO loops which are common in other languages such as Fortran and Pascal. Later, we shall see that it is possible to have DO loops that are used in control statements. These are very useful in running programs multiple times with selected variables changed.

There is, however a block that acts similar to the DO loop but is restricted. This block is the LOOP block and it acts in connection with a transaction's specified parameter. The form of it is:

```
LOOP    A,(block label)
```

where A is a parameter. For example,

```
LOOP    1PH,BACK1
```

would be such a block. The way the loop block works is as follows:

The transaction's parameter as specified in the A operand is evaluated. This is decremented by 1 and the result compared with 0. If the value is zero, the transaction is routed to the next sequential block. If the value is greater than zero, the transaction is routed to the block given by the B operand. This block is always before the LOOP block. Some examples of this block are:

```
a) LOOP    3PH,BACK1
b) LOOP    PH1,UPTOP
c) LOOP    5PF,OVER
```

In a), suppose that the value of the transaction's third half word parameter is 6. Then the looping is done for value of 5, 4, 3, 2, and 1.

In b), first the transaction's first parameter is given. Suppose this is 4 and the value of parameter 4 is 5. Looping is done for 3, 2, and 1.

In c), the looping is done depending on the value of the transaction's parameter number 5.

Looping is done only by a decrement of 1 and cannot be done by increments. As restrictive as this block may seem, there are many uses for it.

The EQU Compiler Directive

You can use parameter numbers as operands of other blocks such as the QUEUE and SEIZE blocks. In fact, this is often done. Thus, one could have blocks such as:

```
QUEUE   PH2
QUEUE   PH1+PH4
SEIZE   PH4
```

Use of such code can greatly compress the number of lines of GPSS. For example, suppose there are three types of ships entering a harbor. Type 1 requires 1 tug boat to berth it, type 2 requires s tug boats and type 3 requires 4 tugboats. Rather than have 3 nearly identical segments, one could have each ship type have a different number in one of its parameters, say parameter 5. Thus, type one ships might have a 1 in parameter 5, type 2 a 2 and type three, a 3. Then you could have:

```
ENTER    TUGS,PH5
```

where TUGS is the storage to represent the number of tug boats available. In addition, you could have the ships enter separate queue by:

```
QUEUE    PH5
```

If you wanted the ships to be in a global queue, it is tempting to write:

```
QUEUE    WAIT
QUEUE    PH5
```

Unfortunately, there is a problem. Assuming there are no other queues in the program, during compiling the queue WAIT is assigned to the first queue. Now, when a type one ship enters queue 1, this is not only QUEUE 1 but also QUEUE WAIT. The statistics will be incorrect. One way around this is to have the queue WAIT renamed as, say, QUEUE 10.

This will work but it is better to use mnemonics that represent names that are more meaningful than just numbers. The EQU compiler allows the analyst to use names for operands but assigns them specific numbers. The general form of this is:

```
(label)  EQU   (number),(entity type)
```

The (entity type) is the family name of the entity. These are:

entity	family name
facility	F
function	Z
parameter	PH,PB,PF or PL
queue	Q
random number	RN
storage	S

The (label) is the mnemonic you want to use. The (number) is the integer number you want to assign to the entity. Some examples of this are:

```
a) WAIT   EQU   7,Q
b) MACH1  EQU   10,F
c) HALTIT EQU   4,Q
```

In a), the queue WAIT is specified as being the seventh queue of all of the possible queues GPSS/H has. In a program if you had the following:

```
QUEUE   WAIT
QUEUE   PH1
```

and PH1 was only 1, 2, or 3, there would be no problems in having the queue WAIT as specified as queue 7. In b), the facility MACH1 is specified as being the tenth facility. In c), the queue HALTIT is specified as being the fourth queue. It is possible to have the following:

```
HALT  EQU     10,Q,F
      ------------
      ------------
      QUEUE   HALT
      SEIZE   HALT
```

Now, both queue HALT and facility HALT are designated at the 10th queue and facility respectively. The next example will illustrate this.

Example 14B.

Three types of ships that use a harbor, type 1, type 2 and type 3. Ships of type 1 and type 2 enter the harbor and are guided into a dock by either 1 or 2 tug boats. Type 1 ships need 1 tug boat and type 2 ships require 2 tug boats. Type 3 ships require 3 tug boats. When ships leave the docks, type 1 and type 2 ships cycle to another port and eventually return. Type 3 ships enter the harbor every 14 ±7 hours and then leave for good. The number of each of type 1 and type 2 and the unloading/loading times for all ships in the harbor are given in Table 14.1. All times are in hours.

Table 14.1 Data Needed for Example 14.2

ship	type number	time in dock	time to cycle
1	12	24 ± 6.6	180 ± 25

2	14	28 ± 8.6	206 ± 27
3		27 ± 7.7	

When a ship enters the harbor, it must wait until a berth is free. There are three berths available. There are 3 tug boats available. When a berth is free, and there are ships waiting in a queue, the first ship in the queue checks to see if enough tug boats are available. If so, the tug boat takes 1 hour to dock it. After unloading/loading the ship again checks to see if enough tug boats are available. If so, it takes .15 hours for the tug boat to move the ship far enough away to free the berth. It then takes 1 ± .2 hours to complete unberthing and free the tug(s).

Build a GPSS/H model to simulate the harbor facility. When the ship transactions are put into the system, have them spaced out by 24 hours each and have the ships of type 1 and type 2 enter the system via the respective ADVANCE blocks so that they are initially at the beginning of their cycle away from the harbor. Form separate queues for each type ship as well as a global queue. The program is to have all the ships in the main segment rather than have 3 separate segments. Simulate for 2 years of 365 days, 24 hours per day of operation.

Solution

The program to simulate the system is given next:

```
      SIMULATE
      STORAGE    S(DOCK),3/S(TUGS),3  PROVIDE BERTHS, DOCKS
WHICH FUNCTION   PH1,D3
1,FIRST/2,SECOND/3,THIRD
WAIT  EQU        10,Q                 DEFINE WAIT AS QUEUE NO. 10
SHIPA GENERATE   ,,,12,,1PH,2PL       PROVIDE TYPE 1 SHIPS
      ASSIGN     1,1,PH               NUMBER THEM 1
      ASSIGN     1,24,PL              MEAN UNLOAD/LOAD TIME
      ASSIGN     2,6.6,PL             SPREAD
      ADVANCE    N(SHIPA)*24          SPACE OUT THE SHIPS
      TRANSFER   ,FIRST               SEND TO SEA
SHIPB GENERATE   ,,,14,,1PH,2PL       PROVIDE TYPE 2 SHIPS
      ASSIGN     1,2,PH               NUMBER THEM 2
      ASSIGN     1,28,PL              MEAN UNLOAD/LOAD TIME
      ASSIGN     1,8.6,PL             SPREAD
      ADVANCE    N(SHIPB)*24          SPACE OUT THE SHIPS
      TRANSFER   ,SECOND              SEND TO SEA
      GENERATE   14,7,,,,1PH,2PL      TYPE 3 SHIPS ARRIVE
```

```
      ASSIGN     1,3,PH                NUMBER THEM 3
      ASSIGN     1,27,PL               MEAN UNLOAD/LOAD TIME
      ASSIGN     1,7.7,PL              SPREAD
HARBOR QUEUE     WAIT                  JOIN GLOBAL QUEUE
      QUEUE      PH1                   JOIN INDIVIDUAL QUEUES
      ENTER      DOCK                  IS A DOCK FREE?
      ENTER      TUGS,PH1              ARE TUG BOATS FREE?
      DEPART     WAIT                  LEAVE THE GLOBAL QUEUE
      DEPART     PH1                   LEAVE INDIVIDUAL QUEUE
      ADVANCE    1                     TUG BOATS BERTHS SHIP
      LEAVE      TUGS,PH1              FREE THE TUG BOATS
      ADVANCE    PL1,PL2               UNLOAD/LOAD A SHIP
      ENTER      TUGS,PH1              ENGAGE TUG BOATS
      ADVANCE    .15                   LEAVE DOCK
      LEAVE      DOCK                  FREE DOCK
      ADVANCE    1,.2                  FINISH UNBERTHING
      LEAVE      TUGS,PH1              FREE TUG BOATS
      TRANSFER   ,FN(WHICH)            TRANSFER SHIPS
FIRST ADVANCE    180,25                TYPE 1 SHIPS AT SEA
      TRANSFER   ,HARBOR               BACK TO HARBOR
SECOND ADVANCE   206,27                TYPE 2 SHIPS AT SEA
      TRANSFER   ,HARBOR               BACK TO HARBOR
THIRD TERMINATE  TYPE                  3 SHIPS LEAVE
      GENERATE   24*365*2              SIMULATE FOR 2 YEARS
      TERMINATE  1
      START      1,NP
      PUTPIC     LINES=10,FILE=SYSPRINT,(S(DOCK)+R(DOCK),_
S(TUGS)+R(TUGS),QA(WAIT),QA1,QA2,QA3,SR(DOCK)/10,SR(TUGS)/10)
|================================================|
|  NUMBER OF DOCKS WAS                      ***  |
|  NUMBER OF TUG BOATS WAS                  ***  |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE   **.**  |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE         **.**  |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE         **.**  |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE         **.**  |
|  UTILIZATION OF DOCK                      **.**%  |
|  UTILIZATION OF TUG BOATS                 **.**%  |
|================================================|
      CLEAR
```

```
      STORAGE    S(TUGS),4
      START      1,NP
      PUTPIC     LINES=10,FILE=SYSPRINT,(S(DOCK)+R(DOCK),_
S(TUGS)+R(TUGS),QA(WAIT),QA1,QA2,QA3,SR(DOCK)/10,SR(TUGS)/10)
|=================================================|
|  NUMBER OF DOCKS WAS                      ***   |
|  NUMBER OF TUG BOATS WAS                  ***   |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE   **.** |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE         **.** |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE         **.** |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE         **.** |
|  UTILIZATION OF DOCK                      **.**% |
|  UTILIZATION OF TUG BOATS                 **.**% |
|=================================================|
      CLEAR
      STORAGE    S(TUGS),3/S(DOCK),4
      START      1,NP
      PUTPIC     LINES=12,(S(DOCK)+R(DOCK),_
S(TUGS)+R(TUGS),QA(WAIT),QA1,QA2,QA3,SR(DOCK)/10,SR(TUGS)/10)
|=================================================|
|  NUMBER OF DOCKS WAS                      ***   |
|  NUMBER OF TUG BOATS WAS                  ***   |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE   **.** |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE         **.** |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE         **.** |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE         **.** |
|  UTILIZATION OF DOCK                      **.**% |
|  UTILIZATION OF TUG BOATS                 **.**% |
|=================================================|
      END
```

The program should be carefully studied. Notice the use of the EQU compiler directive:

```
 WAIT  EQU    10,Q
```

This sets the queue WAIT equal to the 10th queue in the program. If this had not been done, when a type 1 ship executed the block:

```
      QUEUE    PH1
```

It would have also been entering the queue WAIT.

The output from the program is as follows:

```
|=================================================|
|  NUMBER OF DOCKS WAS                        3     |
|  NUMBER OF TUG BOATS WAS                    3     |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE    1.37  |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE          0.36  |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE          0.47  |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE          0.54  |
|  UTILIZATION OF DOCK                       92.01% |
|  UTILIZATION OF TUG BOATS                  28.28% |
|=================================================|

|=================================================|
|  NUMBER OF DOCKS WAS                        3     |
|  NUMBER OF TUG BOATS WAS                    4     |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE    1.20  |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE          0.30  |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE          0.42  |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE          0.48  |
|  UTILIZATION OF DOCK                       90.66% |
|  UTILIZATION OF TUG BOATS                  21.14% |
|=================================================|

|=================================================|
|  NUMBER OF DOCKS WAS                        4     |
|  NUMBER OF TUG BOATS WAS                    3     |
|  AVERAGE NUMBER OF SHIPS IN GLOBAL QUEUE    0.28  |
|  AVERAGE NO. TYPE 1 SHIPS IN QUEUE          0.08  |
|  AVERAGE NO. TYPE 2 SHIPS IN QUEUE          0.11  |
|  AVERAGE NO. TYPE 3 SHIPS IN QUEUE          0.10  |
|  UTILIZATION OF DOCK                       68.96% |
|  UTILIZATION OF TUG BOATS                  28.40% |
|=================================================|
```

The results of the program indicate that with 3 docks and 3 tugs, the docks are used 92.01% of the time and the tug boats are busy 28.28% of the time. The average number of ships waiting in the

global queue is 1.37. When another tug boat is added, the changes are minor: the docks are still busy over 90%, namely, 90.66% and the tug boats are busy 21.14%. The average queue of all the ships is 1.20. But when another dock is added, the changes are quite dramatic: the docks are now used only 68.96% of the time and the tug boats are busy 28.40% of the time. But the average global queue is reduced to .28 ships.

Exercises

1. A transaction has values 1 and 2 in half word parameters 1 and 2 respectively. State what will happen when the transaction enters each block:

```
FIRST FUNCTION   PH1,D2
1,6/2,10
SECOND FUNCTION  PH2,D2
1,2/2,3
      --------
      --------
   a)  ADVANCE    PH1*60
   b)  ADVANCE    FN(FIRST)*FN(SECOND)
   c)  QUEUE      PH2
   d)  ENTER      TUGS,PH1
   e)  LEAVE      SHIP,PH1*PH2
   f)  ADVANCE    FN(FIRST),PH2
   g)  ADVANCE    10,FN(FIRST)
   h)  ASSIGN     3,PH1,PH
       ASSIGN     4,PH2,PH
       ADVANCE    PH3*PH4
   i)  ASSIGN     3,PH1,PH
       ASSIGN     4,PH2,PH
       ADVANCE    PH4,PH3
```

2. Assume that the following entities hold at a particular time in a program.

entity	value
F(MACH1)	1
Q(WAIT1)	2
Q(WAIT2)	3

FR(MACH1)	578
S(TUGS)	5
SC(TUGS)	123
R(TUGS)	2

A transaction has the value of its first 6 half word parameters as follows. All other parameter values are 0.

parameter number	value
1	3
2	-1
3	-2
4	5
5	4
6	6

State what happens if the transaction were to enter each of the following independent blocks:

```
a) ASSIGN   4,F(MACH1),PH
b) ASSIGN   4+,F(MACH1),PH
c) ASSIGN   PH4,F(MACH1),PH
d) ASSIGN   PH4-,F(MACH1),PH
e) ASSIGN   1,PH1*PH5,PH
f) ASSIGN   1-,PH1*PH5,PH
g) ASSIGN   Q(WAIT1),PH3,PH
h) ASSIGN   PH5,FR(MACH1)/S(TUGS),PH
i) ASSIGN   1,Q(WAIT1)+Q(WAIT2),PH
j) ASSIGN   S(TUGS),SC(TUGS),PH
k) ASSIGN   S(TUGS)-PH1,PH6,PH
l) ASSIGN   PH5+,SC(TUGS)-F(MACH1),PH
```

3. A construction job is using a single shovel and 10 trucks, 6 of type 1 and 4 of type 2. The shovel can load only one truck at a time. Each truck is loaded in 1.2 ± .5 minutes. Both type of trucks then travel to a junction in the same time, namely, a time given by the normal distribution with a mean of 5 minutes and a standard deviation of .75 minutes. At the junction, trucks of type 1 travel to a dump area in 2 ± 1.2 minutes. Type 2 trucks travel to a different dump area in 2.5 ± .8 minutes. Both types of trucks take 1 ± .2 minutes to dump. Type 1 trucks return to the shovel in 5.5 ± 2

minutes and type 2 trucks return to the shovel in 3.5 ± 1.2 minutes. Simulate for 5 continuous shifts of 480 minutes each. You are to use the same ADVANCE blocks for both truck types.

4. Consider the following code.

```
      GENERATE    ,,,1
      ASSIGN      1,5,PH
      ASSIGN      5,2,PH
BACK  ASSIGN      2+,PH1,PH
      LOOP        1PH,BACK
      TERMINATE   1
      START       1
```

What will be the value of the transaction's second parameter at the end of the program?

5. Suppose that the LOOP block was LOOP PH1,BACK in exercise 4. What would the value of the transaction's third parameter be at the end of the prkgram?

Solutions

1. a) The transaction is placed on the FEC for 60 time units.
b) The transaction is placed on the FEC for 18 time units.
c) The transaction enter the QUEUE 2
d) The transaction takes one storage from storage TUGS
e) The transaction leaves storage SHIP and frees 2 storages.
f) The transaction is placed on the FEC for a time of 6 ± 2 time units.
g) The transaction is placed on the FEC for a time of 60 time units. Recall that when the B operand of the ADVANCE block is a function, this is multiplied by the A operand.
h) The transaction is placed on the FEC for a time of 2 time units.
i) The transaction is placed on the FEC for a time of 2 ± 1 time units.

2. The parameter types for these are all half word.

a) parameter 4 is 1.
b) parameter 4 is 6.
c) parameter 5 is 1.
d) parameter 5 is 3.
e) parameter 1 is 12.
f) parameter 1 is -9.
g) parameter 5 is -2.

h) parameter 4 is 115 (integer division).
i) parameter 1 is 8.
j) parameter 5 is 123.
k) parameter 2 is 6.
l) parameter 4 is 122.

3. The program to do the simulation is:

```
      SIMULATE
ONE   FUNCTION   PH1,D2
1,2/2,2.5
TWO   FUNCTION   PH1,D2
1,1.2/2,.8
THREE FUNCTION   PH1,D2
1,5.5/2,3.5
FOUR  FUNCTION   PH1,D2
1,2/2,1.2
      GENERATE   ,,,6
      ASSIGN     1,1,PH
      TRANSFER   ,UPTOP
      GENERATE   ,,,4
      ASSIGN     1,2,PH
UPTOP QUEUE      WAIT
      SEIZE      SHOVEL
      DEPART     WAIT
      ADVANCE    1.2,.5
      RELEASE    SHOVEL
      ADVANCE    RVNORM(1,5,.75)
      ASSIGN     3,FN(ONE),PH
      ASSIGN     4,FN(TWO),PH
      ADVANCE    PH3,PH4
      ADVANCE    1,.2
      ASSIGN     3,FN(THREE)
      ASSIGN     4,FN(FOUR)
      ADVANCE    PH3,PH4
      TRANSFER   ,UPTOP
      GENERATE   480*5
      TERMINATE  1
      START      1
```

```
END
```

4. The value would be 15. This is computed as follows: The first time through the ASSIGN 2+,PH1,PH the value of parameter 1 is 5. The next time it is 4, the next time 3, etc. to 1. The sum of 5 + 4 + 3 + 2 + 1 is 15.

5. The value would be 10. This is computed as follows: The first time through the ASSIGN 2+,PH1,PH the value of parameter 1 is 5. But the LOOP block loops on the parameter whose value is specified by parameter 1,not parameter 1. The value of parameter 1 is 5 and so parameter 5 has the value 2. The looping is done with parameter 5 equal to 1 (it is decremented from 2) and after the first loop, the value of parameter 2 is 10 (5 + 5). This is the only time it loops.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul **Mine Design Using Simulation**

Chapter 15
Tables in GPSS/H

It is possible in GPSS to make tables of any SNA. This table might be the length of a queue, the value of a parameter, the percent a machine is working, etc. Often, the time a transaction has been in the system is tabulated or how long it took to go from one point in the program to another point is of interest. GPSS/H makes these tables in the form of histograms. For example, in studying people entering a hardware store and using the shopping carts the simulation might give a maximum number of carts in use as 14. The mean number of carts in use at any time might be 4.5. It would be instructive to see how often 14 carts were in use as well as the distribution of the usage of the carts during the simulation. This can be done easily in GPSS.

In making up the tables, GPSS/H also computes the sample mean of the data, the sample distribution, the number of samples which fall into each of the ranges and the percentage of values in the sample which fall into each of the ranges in the series. This is all done automatically.

Recall that a histogram has intervals which records the number of times a variable falls in each interval. Since a person doing a simulation is often concerned with tabulating data, GPSS provides a very simple method of doing this. In fact, to make a histogram for any SNA takes only two lines of code (!). One of these is a block and the other a statement.

THE TABLE STATEMENT

To record data in a table requires the defining of a statement called the Table Statement. Its general form is:

```
name TABLE SNA, start point, interval width, number of intervals
```

Some examples of the TABLE statement might be:

```
a) FIRST TABLE   S(CARTS),0,1,20
b) MARK1 TABLE   Q(WAIT),0,1,15
c) NEXT  TABLE   FR(MACH1),0,50,22
```

Example a) will record the amount of storage used in the storage CARTS and put them in a table with intervals from 0 to 1. The table will have 20 intervals. The first will go from minus infinity to 0 (even though there will never be an entry here), the second from 0 to 1 and the last end point from

20 to plus infinity. GPSS will count the intervals you specify as well as one before the data and one after. In general, these are the regions from minus infinity to the first data point of the histogram as specified in the B operand and from the last data point of the histogram as specified by the C operand to + infinity. This is important to keep in mind when using the TABLE statement.

The second table will give the distribution of the number of people who were in the Queue called WAIT. The table will have 20 entries which include the number from minus infinity to greater than 20.

The third table gives the utilization of the facility called MACH1. Recall that this is given in parts per thousand. Thus, the table will go from 0 to 50, 50 to 100, 100 to 150 etc., (not counting the end intervals).

THE TABULATE BLOCK

To make an entry in a table, you use the TABULATE block

```
TABULATE  name
```

Every time a transaction enters this block an entry in made in the TABLE with the label name. Thus, the combination of a TABLE statement and the TABULATE block such as:

```
FIRST TABLE     Q(WAIT),0,1,10
..... ..............
..... ..............
      TABULATE  FIRST
```

would make a table (histogram) of the people in the queue named WAIT.

To illustrate what a table looks like from a program, consider the program to study people using a single facility. People arrive every 12 ± 4 minutes. The service time is 11 ± 7 minutes. If the server is busy, people wait in a queue until he is free. Simulate for 50 days straight of 480 minutes per day. Obtain a table of times for people who had to wait for service.

The program to do this simulation is:

```
      SIMULATE
INQUE TABLE     QX(WAIT),0,1,20
      GENERATE  12,4
      QUEUE     WAIT
```

```
      SEIZE      MACH1
      TABULATE   INQUE
      DEPART     WAIT
      ADVANCE    11,7
      RELEASE    MACH1
      TERMINATE
      GENERATE   480*50
      TERMINATE  1
      START      1
      END
```

The block entry count and the table from the output are as follows:

```
BLOCK   CURRENT   TOTAL
1                 1997
2                 1997
3                 1997
4                 1997
5                 1997
6       1         1997
7                 1996
8                 1996
9                    1
10                   1

TABLE INQUE

ENTRIES IN TABLE  MEAN ARGUMENT  STANDARD DEVIATION  SUM OF ARGUMENTS
     1997.0000     10.4626             1.3307       20893.7941 NON-
WEIGHTED

UPPER  OBSERVED  PERCENT  CUMULATIVE  CUMULATIVE  MULTIPLE  DEVIATION
LIMIT  FREQUENCY OF TOTAL PERCENTAGE  REMAINDER    OF MEAN  FROM MEAN
0.        1.0000     0.05       0.05       99.95     0.        -7.8625
1.0000    3.0000     0.15       0.20       99.80     0.0956    -7.1110
2.0000    3.0000     0.15       0.35       99.65     0.1912    -6.3596
3.0000    4.0000     0.20       0.55       99.45     0.2867    -5.6081
4.0000    1.0000     0.05       0.60       99.40     0.3823    -4.8566
5.0000    4.0000     0.20       0.80       99.20     0.4779    -4.1051
```

```
6.0000   50.0000     2.50      3.30      96.70     0.5735    -3.3536
7.0000   16.0000     0.80      4.11      95.89     0.6691    -2.6021
8.0000    9.0000     0.45      4.56      95.44     0.7646    -1.8506
9.0000    3.0000     0.15      4.71      95.29     0.8602    -1.0991
10.0000 268.0000    13.42     18.13      81.87     0.9558    -0.3476
11.0000 887.0000    44.42     62.54      37.46     1.0514     0.4039
12.0000 717.0000    35.90     98.45       1.55     1.1469     1.1553
13.0000  30.0000     1.50     99.95       0.05     1.2425     1.9068
14.0000   1.0000     0.05    100.00       0.00     1.3381     2.6583
```

As shown in the table, there were 1997 entries

There is nothing else to do to make tables of SNA's. This ability to make tables of any SNA so easily and rapidly is often hard for people to believe the first time they are introduced to it.

SNA's Associated with Tables

There are several SNA's associated with tables. These are as follows:

TB(name) or TBj Sample mean. For the previously table, INQUE, this would be 10.4626.

TC(name) or TCj Number of observations. This is 1977 for the table INQUE.

TD(name) or TDj Standard deviation. For the table, INQUE, this would be 1.3307.

THE SNA M1

There is an SNA that is quite useful when constructing tables. This is called M1 and gives the time a transaction has been in the system. Whenever a transaction is created, it is marked with the time it enters the system (using the absolute clock!). The SNA M1 takes the time of the absolute clock and subtracts the entry time from it. This gives the time that the transaction has been in the system. For example, suppose a transaction entered the system at time 404.56 and now the absolute clock is at 625.77, the SNA M1 is equal to 231.21. Consider the following lines of code:

```
TIMES TABLE      M1,0,25,30
      ......
      ......
      TABULATE   TIMES
```

The table TIMES will give the tabulation of times that the transactions are in the system from the time they entered up to the time they encounter the block TABULATE TIMES. The times will be in intervals of 25 and there will be 30 such intervals.

THE MARK BLOCK

Suppose you want a tabulation of times for the transaction to go from point A to point B in a system. It is first necessary to make a record of the time the transaction is at point A. This is done by making a record of the clock time when the absolute clock value in a parameter of the transaction. Which parameter this time is to be stored in must be specified. The block that does this is the MARK block. The general form of the block is

```
MARK P(type)j
```

where (type) can be either PH, PF, PL or PB. "j" is the number of the parameter

note: MARK (number of parameter)$(parameter type) will also work. This form will not be used here.

When a transaction leaves this block, the effect is to put the absolute clock time in the transaction's parameter as specified by the operand number. Since the absolute clock is a real number, if the parameter number does not refer to a floating point parameter it is truncated to an integer. When this happens, the following message appears: "IN STATEMENT 9 - WARNING 393 - Clock value (floating point) will be truncated to an integer value." Thus, if the clock was at 1234.567 and you had:

```
MARK 4PH
```

Half word parameter 4 would have the value of 1234. This is normally not desired and so when using this block, it is recommended that floating point parameters be used to avoid this round off error. For example, you could have:

```
GENERATE    ,,,1,,5PL
.........   .........
MARK        5PL
```

If you had omitted the PL in the MARK block, you would get an error message.

But don't forget that since you are now specifying that the transaction has 5 floating point parameters, it does not have any half word parameter. The SNA that goes with the MARK block is;

```
MP(type)(number)
```

where (number) is the parameter number. The effect of this is to subtract this value from the current absolute clock value. Referring to the same example, if the clock value was now 2344.777, the value of MPL4 would be 1000.210.

Examples of this are:

```
FIRST TABLE      MPL1,0,50,20
SECOND TABLE     MPL2,0,40,22
THIRD TABLE      MPL3,0,30,25
      GENERATE   30,12,,,,12PH,3PL
      ..........
      MARK       1PL
      ..........
      MARK       2PL
      ..........
      MARK       3PL
      TABULATE   FIRST
      TABULATE   SECOND
      TABULATE   THIRD
```

The above will give three tables of times it takes the transactions to travel from the three points in the program where it has passed through the blocks

```
      MARK       1PL
      MARK       2PL
      MARK       3PL
```

ADDITIONAL TABLES

There are several tables that GPSS can make for you with only a minor change to the program. These tables are as follows:

IA MODE TABLE

Whenever a GENERATE Block is used, the distribution of interarrival times is required as data. Often, the interarrival times at points interior to a model are required. The logic of doing this is

clear. At a particular point in a model, whenever a transaction arrives, a record is made of the arrival time, say, 456.78 time units. At a later time a second transaction arrives at the same point at, say, 666.99 time units. The interarrival time is then determined as: 666.99 - 456.78 or 210.21.

This time can be tabulated by the following lines of code:

```
FIRST TABLE      IA,0,20,25
      ..........
      TABULATE   FIRST
```

The IA in the TABLE statement is essential for the first operand.

RT MODE TABLE

Closely related to the IA mode is the rate of the arrivals at a point. For example, the arrivals per 10 seconds, the arrivals per minute, etc. A table having these values is constructed as follows:

```
SECOND TABLE     RT,0,15,25,10
```

Notice that there is an "E" operand in the TABLE statement. This gives the time span to be used. In the above example, the table will give arrivals per 10 time units. To make an entry in the table, the Block

```
      TABULATE  SECOND
```

is used.

Q-TABLE MODE

The average residence time in a queue is often required. The average time is given in the ordinary output. The Q-Table gives the table of average times in the queue. Amazingly enough, this table requires only a single line of code:

```
NAME  QTABLE    WAIT,0,600,20
```

Whenever a transaction leaves the queue WAIT, an entry is made in the table of the time it spent in the queue.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 16
SAVEVALUES

Up to this point all of the SNA's we have used were supplied by either one of the blocks or internal to GPSS. Often the programmer will want to have his or her own user supplied SNA's. Parameters have been used to store various different numbers but these are not printed out after a program is run.

GPSS provides for user defined SNA's and calls them savevalues (values that are "saved") These values are printed out in the output report, except for zero values. GPSS/H has 4 different types of these savevalues. They are:

1. **Full word savevalues. These are integers, which range from -2,147,483,648 to +2,147,483,647. Their family name is XF. A full word savevalue is referenced by XF(name) or simply X(name). Alternately, one can use a single dollar sign "$" such as X$name. This is an old way of referencing savevalues which still works in GPSS/H. A savevalue can be a number. In that case, it is referenced simply by XF_j or X_j.**
2. **Half word savevalues. These are integers, ranging from -32,768 to +32,767. Their family name is XH.**
3. **Bit word savevalues. These are integers, ranging from -128 to +127. Their family name is XB.**
4. **Floating point savevalues. These are decimal values whose family name is XL. Their size depends on the computer.**

To reference a savevalue, it is necessary to do so using the family name with parenthesis. Thus,

```
ADVANCE     XF(SPEED)
GENERATE    XL(AVER),XL(SPREAD)
QUEUE       XH(WAIT)
ASSIGN      1,XL(FIRST),PL
SEIZE       XB(MACH1)
```

are all examples of using savevalues as operands of other blocks. If the second letter in the family name is omitted, it is assumed that the savevalue is a fullword savevalue. For example,

```
ASSIGN 1,XF(TEST),PH
```

and

```
ASSIGN 1,X(TEST),PH
```

are the same. In both cases, the value of the savevalue TEST will be given to the transaction's first half word parameter. In this text, the exact type of the savevalue will be fully specified. Thus, reference to savevalues as X(TEST) will not be done.

It is also possible to reference a savevalue using the dollar sign, $. XL$NUMB and XL(NUMB) are the same. This method of referencing savevalues will not be used here as it is a holdover from the original versions of GPSS.

How to make changes to the values of savevalues is given next.

The SAVEVALUE Block

All savevalues are initially zero. **How to give them initial values different from zero will be considered shortly. During the running of a program, it is often desired to change their values. This is done by the SAVEVALUE block. Its general form is:**

```
SAVEVALUE (name),value,type
```

where (name) is the name of the savevalue. These names are assigned according to the usual rules. This could be a number or any SNA.

value is the value to be assigned to the savevalue. This could be a SNA.

type is the family name and is either XF, XH, XB or XL. If this specification is omitted, the savevalue is a full word savevalue by default.

Some examples of the SAVEVALUE block are:

```
a) SAVEVALUE JIM,2,XF
b) SAVEVALUE TOMMY,-100,XH
c) SAVEVALUE JOE,32,XF
d) SAVEVALUE 1,4,XF
e) SAVEVALUE FIRST,25.63,XL
f) SAVEVALUE NEXT,25/2,XL
g) SAVEVALUE OTHER,25.0/2,XL
h) SAVEVALUE PH4,12,XH
```

In a), the savevalue JIM is set equal to 2. It is a full word savevalue.

In b), the savevalue TOMMY is set equal to -100 and it is a half word savevalue.

In c), the savevalue JOE is set equal to 32.

In d), the savevalue "1" is set equal to 4. It, too, is a full word savevalue.

In e), the savevalue FIRST is set equal to 25.63.

In f), the value of NEXT is set equal to 12.0000. It is not 12.5000 because division is integer division unless specified by using floating point numbers.

In g), the division is floating point division because of the decimal in the 25.0 (GPSS/H converts the 2 to a floating point number). Here the value of OTHER is 12.5000. (The reason for the 4 decimals is because this is what GPSS/H prints out for the value of the savevalues, not because this is the number of digits actually stored).

In h), it is not known what savevalue will have the value of 12. This will depend on the number stored in the transaction's 4th half word parameter. If it is 5, then savevalue 5 will have the value 12.

The use of a number for the value of a variable may seem confusing as in other languages variables normally must be alphanumeric starting with a letter. For example, in Fortran, one might have statements defining variables such as: JIM = 4, TOMMY = -100.0, JOE = 32 and X1 = 1, but not 2 = 1. In effect, these do the same thing as the first four examples of the SAVEVALUE block given above. Corresponding Fortran statements for examples e), f) and g) might be: FIRST = 25.63, NEXT = 25/2 and OTHER = 25.5 .

The use of numbers for savevalues will be avoided here if at all possible. There will be an example when this feature will come in very handy, especially when using a variable for the first operand of the savevalue. For example, consider the SAVEVALUE:

```
SAVEVALUE   PH1,3
```

Here the savevalue to be given the value of 3 will depend on the transaction's first parameter. If it happened to be 4, then the savevalue 4 is given the value 3.

The FIX and FLT Mode Conversion

Suppose you want to have the value of savevalue FIRST to be equal to the sum of Q(ONE) plus

Q(TWO) to be divided by Q(THREE) and you want an exact floating point result. It would not be correct to write:

```
SAVEVALUE  FIRST,(Q(ONE)+Q(TWO))/Q(THREE),XL
```

Since the queue lengths of queue is an integer, the quotient will represent integer division. In order to have floating point division, one could do the following:

```
SAVEVALUE  JUNK1,Q(ONE)+Q(TWO),XL
SAVEVALUE  JUNK2,Q(THREE),XL
SAVEVALUE  FIRST,XL(JUNK1)/XL(JUNK2),XL
```

This is a bit awkward. GPSS/H offers two mode conversion operators. These are FIX and FLT to convert to fixed point or floating point mode. They work identically to the mode converters found in other languages such as Fortran where the corresponding mode converters are IFIX and FLOAT. In GPSS/H, one might have:

```
FLT(Q(ONE))
FIX(XL(TEST))
FLT(FC(MACHA))
```

Thus, one could have written the original expression for the savevalue as:

```
SAVEVALUE  FIRST,(Q(ONE)+Q(TWO))/FLT(Q(THREE)),XL
```

Notice that it was not necessary to convert all three fixed point queue values. Savevalues can be used in increment or decrement mode, just as the ASSIGN block was used. Thus,

```
a) SAVEVALUE LOAD+,25,XF
b) SAVEVALUE COST-,XF(PRICE),XF
c) SAVEVALUE PILE+,FN(TRUCK),XF
d) SAVEVALUE PH1+,PH2,XF
e) SAVEVALUE NEXT+,FN(LAST)+FN(FIRST),XL
f) SAVEVALUE TOM-,Q(ONE)/3,XL
g) SAVEVALUE TOM-,Q(ONE)/3.0,XL
```

In a), the savevalue LOAD is incremented by 25.

In b), the savevalue COST is decremented by whatever the savevalue PRICE is.

In c), the savevalue PILE is incremented by reference to the function TRUCK.

In d), the savevalue specified by the transaction's first half word parameter is incremented by the value in its second half word parameter.

In e), the value of the savevalue is given by reference to the functions LAST and FIRST. This is added to the current value of the savevalue NEXT.

In f), the value of the savevalue TOM is decremented by the length of the queue ONE divided by 3 (integer division!).

In g), the savevalue TOM is decremented by the length of the queue ONE divided by 3.0 floating point division). In the first 4 cases, the savevalues are full word savevalues. In the other 3, the savevalues are floating point. In a language such as Fortran, corresponding statements might be:

```
LOAD = LOAD + 25
COST = COST - PRICE
PILE = PILE + F(TRUCK)
X(1) = X(1) + X(2)
NEXT = NEXT + F(LAST) + F(FIRST)
TOM = TOM - QONE/3
TOM = TOM - QONE/3.0 .
```

A common error in programming is to omit the family name XF, XH, XL or XB with the savevalue in parenthesis when referencing a savevalue. If this is omitted, the value of the savevalue is taken to be zero no matter what it actually is. This type of error is most insidious as it can be extremely hard to detect and no run time error takes place. Thus, if you had intended to write:

```
TEST E    XF(VALUE),1,AWAY
```

but instead wrote:

```
TEST E    VALUE,1,AWAY
```

The test will always be false.

The INITIAL Statement

As indicated, the value of all savevalues are set equal to zero when a program begins. This is done by the processor. Often, a programmer will want savevalues to be initially set to non-zero values.

This is done with the INITIAL statement. The general form of it is:

```
INITIAL   Xi(SVj),value/Xi(SVk)/...
```

where X_i is the family name (either XF, XH, XF, XB or XL). SV_j, SV_k, etc. are the names of the savevalues. In case the savevalue name is a number, there is no parenthesis.

Some examples of this are:

```
INITIAL   XF(FIRST),100/XH(TEST),-340/X1,10000
```

An alternate way to write the above is with dollar signs, $:

```
INITIAL   X$FIRST,100/XH$TEST,-340/X1,10000
```

This is a holdover from the early days of GPSS. A shorthand form can be used for multiple initialization as follows:

```
INITIAL   XH1-XH10,3/XH(PLACE),125/XL(TOWN),1234.5432
```

This sets the half word savevalue 1 through 10 equal to 3. It then sets the savevalues PLACE and TOWN equal to the value of the savevalue CITY + 3. For students who have studied Fortran, the INITIAL statement in GPSS is analogous to the DATA statement.

Effect of RESET and CLEAR on Savevalues

A RESET statement does not effect the values of the savevalues but the CLEAR statement sets all savevalue to zero. If the program is to be run again with the original initialized values, there are two things that can be done:

1. Re-initialize all the values with a new INITIAL statement (or statements).
2. Use a form of the CLEAR statement called the selective CLEAR.

The selective CLEAR is simply the CLEAR statement followed by a list of savevalues that are not to be set equal to zero. Thus,

```
CLEAR   XH(TOM),XF(JOHN),XH(PLACE),XF7
```

will clear all the savevalues in the program except for TOM,JOHN, PLACE and 7.

Example 16.1

A common problem in inventory control is known as the "newspaper boy's problem". This concerns a newsboy who sells his papers in the street corner as opposed to one who delivers papers from house to house. The demand for newspapers is uncertain each day but by building up past records the newsboy can determine a probability distribution of expected sales and probability. The newsboy must go in the morning to the main newspaper office and purchase papers. If he does not have enough to sell (demand greater than supply), he must buy papers from a newsstand. These cost him more than if he were able to purchase them himself in the morning but he still makes a profit by doing so. If he has too many, the main office will purchase his unsold stock for a token amount.

Suppose the following data held:

cost / paper:	$.36
selling price:	.55
cost per paper later in the day:	.45
refund per unsold paper:	.15

The expected sales are given by the following table:

Table 16.1. Data from past sales of newspapers

no. papers sold	prob.
50	.05
55	.08
60	.14
65	.20
70	.15
75	.13
80	.10
85	.08
90	.05

95	.02

Determine the amount of papers the newsboy should obtain each day from the head office to maximize his expected profit each day. Simulate for 200 days of sales. Use supply amounts from 50 to 95 in increments of 5. Use the PUTPIC statement to print out only the amount of papers supplied and the resulting expected profit each day. Use a DO LOOP to run the program multiple times. The program to do the simulation is given below:

```
      SIMULATE
      INTEGER    &I                DEFINE AMPERVARIABLE
      RMULT      123               RANDOM NUMBER SEED
SELL  FUNCTION   RN1,D10           HOW MANY TO SELL?
.05,50/.13,55/.27,60/.47,65/.62,70/.75,75/.85,80/.93,85/.98,90/1,95
      GENERATE   ,,,1              DUMMY TRANSACTION
UPTOP ASSIGN     1,FN(SELL),PH     DETERMINE SALES FOR DAY
      TEST GE XH(SUPPLY),PH1,DOWN  IS THIS GREATER THAN SUPPLY?
*
* IF SUPPLY GREATER THAN DEMAND, DETERMINE PROFIT
*
      SAVEVALUE  PROFIT+,19*PH1-(XH(SUPPLY)-PH1)*15,XL
      ADVANCE    1                 ONE DAY PASSES
      TRANSFER   ,UPTOP            BACK FOR ANOTHER DAY
*
* BELOW IS FOR THE CASE THAT THE SUPPLY IS NOT ENOUGH
*
DOWN  SAVEVALUE  PROFIT+,55*PH1-36*XH(SUPPLY)_
-45*(PH1-XH(SUPPLY)),XL
      ADVANCE    1                 ONE DAY PASSES
      TRANSFER   ,UPTOP            BACK FOR ANOTHER DAY
      GENERATE   200               SIMULATE FOR 200 DAYS
      SAVEVALUE  MONEY,XL(PROFIT)/200,XL  DETERMINE AVERAGE PROFIT
      TERMINATE  1                 END OF SIMULATION
      DO &I=1,10                   DO LOOP
      CLEAR                        CLEAR FOR NEXT RUN
      RMULT 123                    RANDOM NUMBER SEED
      INITIAL    XH(SUPPLY),5*&I+45  SET SUPPLY AMOUNT
      START      1                 BEGIN PROGRAM
      END
```

From the output of the program the following table was constructed:

Table 16.2 Results of running the program

no. papers to stock	expected profit (cents)
50	1142
55	1180
60	1210
65	1222*
70	1207
75	1176
80	1127
85	1067
90	1000
95	929

As can be seen from the table, the number of papers to stock each day is 65. This will result in an expected profit per day of $12.22.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul

Chapter 17
THE LOOP block, LOGIC SWITCHES and GATES

The LOOP block

In general, GPSS/H does not allow looping in blocks, but, as we shall see, you can use DO LOOPS for control statements. However, there is one sequence of statements where a form of a DO LOOP is allowed. Consider the lines of code:

```
UPTOP ..........
      ..........
      ..........
      ASSIGN    2-,1,PH
      TEST E    PH2,0,UPTOP
```

The effect of these statements is to create a looping condition between the blocks TEST and the one labeled UPTOP. The loop will continue until half word parameter 2 of the transaction is reduced to zero. If, for example, the value of the parameter was initially 8, the statements between the one labeled UPTOP and the TEST block would be executed 8 times. Such looping arises quite often in GPSS and there is a block that can be used to do exactly this type of looping and so avoid using the inefficient TEST block. The block to do the looping is the LOOP block and the general form of it is:

```
LOOP parameter_no,(label)
```

Some examples of it are:

a) `LOOP 3PH,BACK1`
b) `LOOP 1PH,UPTOP`
c) `LOOP PH4,AGAIN`

The effects of using a LOOP block are:

1) The value of the parameter is decremented by 1. If the parameter is initially negative, an error results.
2) The new value of the parameter is tested with 0.

3) If the value is > 0, the transaction is routed to the block with the label (label).
4) If the value is = 0, the transaction goes to the next sequential block.

No other form of looping is allowed in GPSS blocks. While this may seem highly restrictive, it is surprising how many times the LOOP block can be used.

For the three examples given above:

In a), the loop will be done 3 times for the blocks between the LOOP block and the block labeled BACK1.

In b), the loop will be done only 1 time between the LOOP block and the block labeled UPTOP.

In c), the looping will depend on the value of the transaction's 4th half word parameter. If it is 5, then the loop will be done 5 times; if it is 6, 6 times, etc.

If the transaction has only half word parameters, the LOOP block can be written as:

```
      LOOP   (number),(label)
```

Thus

```
      LOOP   3,BACK1
```

and

```
      LOOP   3PH,BACK1
```

would be identical.

LOGIC SWITCHES

Consider the following blocks

```
      TEST E     XH(LOCK),0
      ......... .........
      ......... .........
      GENERATE   ,,,1
BACK  ADVANCE    RVEXPO(1,125)
```

```
SAVEVALUE   LOCK,1,XH
ADVANCE     RVEXPO(1,12.5)
SAVEVALUE   LOCK,0,XH
TRANSFER    ,BACK
```

When the program begins, the value of the half word SAVEVALUE LOCK is 0 (as are all savevalues unless the INITIAL statement is used to specify initial values) so the transactions that enter the TEST block will pass through to the next sequential block. The transaction created in the GENERATE block is put on the FEC chain for a time given by sampling from the exponential distribution with a mean of 125. After this transaction is returned to the CEC, the value of the SAVEVALUE LOCK is set equal to 1 (any other value could have been selected for the SAVEVALUE for this example). The transaction is then put on the FEC for a time given by sampling from the exponential distribution with a mean of 12.5. Now, any transactions which enter the TEST block will be delayed until the value of the SAVEVALUE LOCK becomes equal to 0. This delay might represent a traffic light turning red, a break in a factory for lunch, a breakdown of an assembly line, etc.

Rather that use SAVEVALUE's and TEST blocks in this manner, there is a better way to handle such conditions in GPSS. This is by using switches that are either "on" or "off". The "on" condition is known as "set" and the "off" as "reset". These switches are turned on and off by the LOGIC block. Its form is as follows:

```
LOGIC (R) (name or number of switch)
```

where R is a logic relationship and is either:

S for set
R for reset
I for invert

When a transaction enters a logic block, the effect is as follows:

a) if the logic relationship is R, the switch is put in a reset position.
b) if the logic relationship is S, the switch is put in a set position.
c) if the logic relationship is I, the switch is inverted, i. e., if it was set, it becomes reset and vice versa.

Examples of these might be:

```
LOGIC S   HALT
LOGIC R   1
```

```
      LOGIC I    WAIT
```

Notice that logic switches can be named either symbolically or numerically. The usual rules apply for selecting names or numbers for these switches.

When a program begins, all switches are in a reset position. It is possible to have the switches in a set position by means of the initial statement. A form of it is as follows:

```
      INITIAL    LS(switch1)/LS(switch2/LS1
```

An example of it is:

```
      INITIAL    LS(HALT)/LS(PATH)/LS1
```

Alternately, in place of the parenthesis, one can use a single dollar sign, $. The above line of code could have been written:

```
      INITIAL    LS$HALT/LS$PATH/LS1
```

If you have multiple switches all given by numbers, there is a shorthand way to have them in a set position:

```
      INITIAL    LS1-5/LS9-12
```

This would put logic switches 1 through 5 in a set position and also switches 9 through 12 in a set position. Some sample program code might be:

```
      GENERATE   ,,,1              DUMMY TRANSACTION
UPTOP ADVANCE    RVEXPO(1,200)     MACHINE WORKING
      LOGIC S    STOPIT            MACHINE DOWN
      ADVANCE    RVNORM(1,20,3.5)  MACHINE BEING FIXED
      LOGIC R    STOPIT            MACHINE FIXED
      TRANSFER   ,UPTOP            BACK TO WORK
```

The above code might be used to represent a machine that works for a certain time and then is shut down where repairs and/or maintenance is performed. The time between breakdowns is given by sampling from the exponential distribution with a mean of 200. When it is down, it is fixed or otherwise maintained. This takes a time that is normally distributed with mean 20 and standard deviation 3.5. Notice how the dummy transaction keeps looping in the program segment and alternately sets the logic switch from set to reset. Although it would not be as clear, it would have

been all right to have the LOGIC blocks as

```
LOGIC I    STOPIT.
```

The GATE Block

Logic switches generally need another block in the main program in order to be of any use. This block is often the GATE block. Logic switches can be used in other blocks such as the TEST block, as we shall see later. This block works as its name implies, much like a gate in the path of the transactions. When the gate is open, transactions pass through and when it is closed, they will either wait until the gate becomes open or be routed elsewhere.

There are two forms of the GATE block. The first is refusal mode. The general form is:

```
GATE R   (logic switch)
```

The logical relation R is either LS or LR. LS stands for "logic switch set" and LR for "logic switch reset". An example of this might be:

```
GATE LS   HALT
```

When a transaction arrives at this block, it tests to see if the logic switch HALT is in a set position. If so, it moves to the next sequential block. If the logic switch HALT is in a reset position, the transaction remains in the previous block. It also remains on the CEC. However, an internal flag is turned to the "on" position and the transaction is not scanned again until such time as the switch is turned to an "off" position. This happens when the LOGIC block is entered by another transaction. If a machine is to be periodically shut down, one might use a GATE block as follows:

```
QUEUE      WAIT
GATE LR    STOPIT
SEIZE      MACH1
DEPART     WAIT
ADVANCE    RVEXPO(1,25)
RELEASE    MACH1
```

Whenever the logic switch STOPIT is in a set position, the transaction is kept in the block QUEUE WAIT. GATE blocks can also be used with facilities and storages as follows:

```
GATE R     label
```

where R can be one of the following:

U test for facility in use
NU test for facility not in use
SF test for storage full
SNF test for storage not full
SE test for storage empty
SNE test for storage not empty

Some examples are:

a) `GATE U MACH1`
b) `GATE SNE TUGS`
c) `GATE NU SHOP`
d) `GATE SF BERTH`

In a), the transaction will be held in the previous block if the facility MACH1 is not in use.

In b), the transaction is delayed if the storage TUGS is in any situation other than empty. Thus, if the storage of TUGS is 4 and one is being used, the transaction will be held up.

In c), the transaction is delayed if the facility SHOP is being used.

Finally, for d), the transaction is held up if the storage BERTH is not full.

The GATE Block in Conditional Transfer Mode

When a GATE block has a second operand, this is the label of a block. If the GATE is closed, the transaction will be transferred to the block with this label. Thus,

```
GATE LR   HALTIT,AWAY
```

will send the transaction to the block AWAY whenever the logic switch HALTIT is in a set position.

Example 17.1

A contractor is excavating for a large shopping center. He has 5 trucks which haul excavated dirt away to a dump. There is a single shovel to load the trucks. The trucks travel in a cycle: load,

haul, dump and return. All times for these activities are found to be normally distributed. Loading times have a mean of 2.5 minutes and a std. dev. of .35 minutes. Travel to the dump has a mean of 7.5 minutes and a std. dev. of 1.26 minutes. Dumping takes an average of 1.75 minutes with a std. dev. of .2 minutes and returning to the shovel takes an average of 5.6 minutes, std. dev. of 1.1 minutes. Only one truck can be loaded at a time but there is no such restriction on dumping. The various times for the trucks are the same regardless of the truck.

The trucks periodically break down and/or must be serviced. Each has a different reliability. Calling the truck 1, 2, 3, 4, and 5, it has been found that the down times for the trucks follow the exponential distribution. The time to repair a truck follows the normal distribution. Table 17.1 gives these times:

Table 17.1 Down times and repair times for the trucks

truck no.	down time dist.	repair time dist
1	400	(20,3.5)
2	425	(35,6.9)
3	550	(55.5,12)
4	345	(40,7)
5	300	(50,8)

The down times are means for the exponential distribution and the repair times are (mean, std. dev.) for the normal distribution. Write the GPSS program to simulate the situation using GATE blocks and LOGIC switches to cause the trucks to be down periodically. Even though the breakdowns of the trucks can be any place in the system, it is sufficient to have the trucks tested for breakdowns after they dump.

Solution

The program is written using half word parameter 1 to store a number from 1 to 5 to represent the 5 different trucks. When a truck dumps, it is sent to one of 5 different GATE blocks depending on what type truck it is. If a truck is down, the corresponding LOGIC switch will be in a set position and it will be held up at this point (the transaction will remain in the TRANSFER block). There are 5 program segments to alternately put the logic switches in set and reset positions. Because the segments are nearly identical, macros are used for both the GATE block segments and the reliability segments. Notice how the two segments work. For example, for the first truck, the

lines of code for it when it is tested to see if it is down and the segment to shut it down are written side by side:

```
BLOCKA BUFFER                        GENERATE   ,,,1,10
       GATE LR   FIRST       BACK1   ADVANCE   RVEXPO(1,400)
       TRANSFER  ,DOWN       TIMEA   LOGIC S    FIRST
                                     TEST E     W(BLOCKA),1
                                     ADVANCE    RVNORM(1,5.6,1.1)
                                     LOGIC R    FIRST
                                     TRANSFER   ,BACK1
```

When a truck transaction is to be tested to see if it is down, it first goes a BUFFER block. This causes a re-scan of the current events chain. The dummy transaction that will alternately set and reset the logic switch FIRST has a priority of 10. Thus, in case of a time tie, it will be moved forward first. Let us suppose that the dummy transaction has left the ADVANCE RVEXPO(1,400) block so that the truck is to be down. The switch FIRST is placed in a set position. However, before the transaction can enter the ADVANCE RVNORM(1,5.6,1.1) block to represent the truck being down, it is held up in the block TEST E W(BLOCKA),1. This is so that the truck will, indeed, be delayed. For example, suppose the truck breakdown occurred just after the truck was loaded and that this breakdown is to be for 8 minutes and suppose that the truck will take 9 minutes to reach the block BLOCKA. If the dummy transaction was not delayed, it would immediately enter the ADVANCE RVNORM(1,5.6,1.1) block and so when the truck arrived at the block BLOCKA, the logic switch FIRST is no longer in a set position and the truck would not experience any delay.

The effect of testing for failures in this manner introduces a slight error in that the trucks will continue to operate when they are to be down until they reach the various GATE blocks to delay them. This can be overcome in several ways: have more such GATE blocks in the program. This will tend to increase the lines of code. Alternately, one can adjust the statistical distributions to compensate for the error introduced. The program listing is a follows:

```
       SIMULATE
FMACRO STARTMACRO
#A     BUFFER
       GATE LR     #B
       TRANSFER    ,DOWN
       ENDMACRO
SMACRO STARTMACRO
       GENERATE    ,,,1,10
#A     ADVANCE     RVEXPO(1,#B)
#C     LOGIC S     #D
```

```
      TEST E     W(#E),1
      ADVANCE    RVNORM(1,#F,#G)
      LOGIC R    #D
      TRANSFER   ,#A
      ENDMACRO
TYPE  FUNCTION   PH1,L5
1,BLOCKA/2,BLOCKB/3,BLOCKC/4,BLOCKD/5,BLOCKE
TIMES GENERATE   ,,,5,5 5           TRUCKS WORKING
      ASSIGN     1,N(TIMES),PH      NUMBER TRUCKS 1,2,3,4,5
UPTOP QUEUE      WAIT               JOIN THE QUEUE
      SEIZE      SHOVEL             USE THE SHOVEL
      DEPART     WAIT               LEAVE THE QUEUE
      ADVANCE    RVNORM(1,2.5,.35)  LOAD A TRUCK
      RELEASE    SHOVEL             FREE THE SHOVEL
      ADVANCE    RVNORM(1,7.5,1.2)  TRAVEL TO DUMP
      ADVANCE    RVNORM(1,1.75,.2)  DUMP
      TRANSFER   ,FN(TYPE)          WHICH TYPE TRUCK?
FMACRO MACRO     BLOCKA,FIRST
FMACRO MACRO     BLOCKB,SECOND
FMACRO MACRO     BLOCKC,THIRD
FMACRO MACRO     BLOCKD,FOURTH
FMACRO MACRO     BLOCKE,FIFTH
DOWN  ADVANCE    RVNORM(1,5.6,1.1)  RETURN TO SHOVEL
      TRANSFER   ,UPTOP             BACK FOR ANOTHER LOAD
SMACRO MACRO     BACK1,400,TIMEA,FIRST,BLOCKA,20,3.5
SMACRO MACRO     BACK2,425,TIMEB,SECOND,BLOCKB,35,6.9
SMACRO MACRO     BACK3,550,TIMEC,THIRD,BLOCKC,55,10
SMACRO MACRO     BACK4,344,TIMED,FOURTH,BLOCKD,40,7
SMACRO MACRO     BACK5,300,TIMEE,FIFTH,BLOCKE,50,8
      GENERATE   480*100
      TERMINATE  1
      START      1,NP
      PUTPIC     LINES=13,FC(SHOVEL)/100,_
FR(SHOVEL)/10,N(TIMEA),N(TIMEB),N(TIMEC),_
N(TIMED),N(TIMEE)
|=========================================|
|   <<RESULTS OF SIMULATION FOR 100 DAYS>>  |
|                                           |
|  NUMBER OF LOADS DUMPED EACH DAY   ***    |
```

```
|  UTIL OF SHOVEL                          ***.**% |
|                                                  |
|  NUMBER OF TIMES TRUCK 1 DOWN            ***     |
|  NUMBER OF TIMES TRUCK 2 DOWN            ***     |
|  NUMBER OF TIMES TRUCK 3 DOWN            ***     |
|  NUMBER OF TIMES TRUCK 4 DOWN            ***     |
|  NUMBER OF TIMES TRUCK 5 DOWN            ***     |
|                                                  |
|==================================================|
     END
```

The output is:

```
|==================================================|
|   <<RESULTS OF SIMULATION FOR 100 DAYS>>         |
|                                                  |
|  NUMBER OF LOADS DUMPED EACH DAY   121           |
|  UTIL OF SHOVEL                     63.11%       |
|                                                  |
|  NUMBER OF TIMES TRUCK 1 DOWN      119           |
|  NUMBER OF TIMES TRUCK 2 DOWN      117           |
|  NUMBER OF TIMES TRUCK 3 DOWN       60           |
|  NUMBER OF TIMES TRUCK 4 DOWN      130           |
|  NUMBER OF TIMES TRUCK 5 DOWN      138           |
|                                                  |
|==================================================|
```

Exercises 17

1. In the Example 17.1 the shovel was busy only 63.11% of the time. Assume that the trucks never failed or if one did a replacement was immediately available. Determine the utilization of the shovel under this condition.

Return on CONTENS

Designed by Vyacheslav V. Franchuk

e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul

Chapter 18
Other forms of the TRANSFER block

In Chapter 4 three forms of the TRANSFER block were discussed. These are the: unconditional TRANSFER, conditional TRANSFER and the TRANSFER BOTH modes. The first two of these are by far the most commonly used. However, there are other forms of the TRANSFER block that can come in quite handy when they are needed. Each will be discussed here with possible applications.

1. THE TRANSFER PICK MODE

This form of the TRANSFER block will select a block to transfer the transaction at random from a number of possible blocks. The transaction will unconditionally go to this block. Each of the blocks to be selected will have the same probability of being selected. Thus, if there are three blocks, each will be selected with probability .3333; for 4 blocks the probability is .250, etc.

The form of the block is:

```
      TRANSFER   PICK,LOCNA,LOCNB
LOCNA .........
      .........
      .........
LOCNB .........
```

LOCNA and LOCNB are block labels. The word PICK is in operand position A. LOCNA must be at a location before LOCNB. Each block between LOCNA and LOCNB is considered in the range of the transfer. This restriction means that, in general, only TERMINATE and TRANSFER blocks are in the range LOCNA and LOCNB. Consider the code:

```
      TRANSFER   PICK,FIRST,LAST
FIRST TRANSFER   ,MACH1
      TRANSFER   ,MACH2
      TRANSFER   ,MACH3
      TRANSFER   ,MACH4
LAST  TRANSFER   ,MACH5
```

A transaction will be transferred to one of the blocks labeled MACH1, MACH2, MACH3,

MACH4 and MACH5 with equal probability.

Example 18.1

What will the following program do?

```
      SIMULATE
      GENERATE    1
      ADVANCE     1
      TRANSFER    PICK,FIRST,LAST
FIRST TERMINATE
      TERMINATE
      TERMINATE
      TERMINATE
      TERMINATE
LAST  TERMINATE
      GENERATE    1000
      TERMINATE   1
      START       1
      END
```

Solution

The program will generate a transaction every 1 time unit. The transaction will be put on the FEC for 1 time unit. When it returns to the CEC, it will then be transferred to one of 6 TERMINATE blocks with equal probability. Selected output from running the program for the 999 transactions that enter the TRANSFER block are:

```
RELATIVE CLOCK: 1000.0000 ABSOLUTE CLOCK: 1000.0000

BLOCK   CURRENT   TOTAL   BLOCK   CURRENT   TOTAL
1                   999   11                    1
2               1   999
3                   998
FIRST               177
5                   164
6                   146
7                   175
```

```
8                170
LAST             166
10                 1
```

Each of the TERMINATE blocks between FIRST and LAST can be expected to be entered approximately 166 times.

2. THE TRANSFER ALL MODE

This form will attempt to transfer the transaction to a series of blocks by trying each one in sequence. For example, if the blocks are FIRST, SECOND, THIRD, LAST, the transaction attempts to first enter the block labeled FIRST. If it can, it does so. If not, it tries the block labeled SECOND, etc. If all of the blocks are in refusal mode, the transaction waits until the first block in the series is free.

The form of the block is:

```
        TRANSFER   ALL,AAAA,BBBB,4
AAAA    --------
        --------
        --------
        --------
CCCC    --------
        --------
        --------
        --------
DDDD    --------
        --------
        --------
        --------
BBBB    --------
```

The word ALL must be in position A, The operands AAAA and BBBB are block labels. Operand D is a positive integer. The 4 shown in operand position D means that the blocks to be transferred to are 4 program lines (blocks) apart. Consider the program outline given above. The transaction first attempts to enter the block labeled AAAA. If it can do so, it does. If not, it attempts to enter the block with the label CCCC. This block is not given in the TRANSFER block but is 4 blocks down from the block AAAA. (The labels CCCC and DDDD need not be in the program). This attempt to enter a block is continued until all blocks 4 apart are considered up to and including block BBBB. If all are in refusal mode, the transaction is held in the TRANSFER block until future

scans of the CEC and one of the blocks becomes able to accept the transaction.

Example 18.2

A factory has three machines to do work on failed parts. Parts arrive every 12 ± 4.5 minutes. Machine A is the best and can finish a part every 14 ± 4.3 minutes. If this machine is free, the part goes there; else it is sent to machine B, which works in 20 ± 5.7 minutes. If both machines A and B are busy, it is sent to machine C which is quite slow, working at 30 ± 3.4 minutes. If machine A is busy, the part is sent to machine B, if A and B are busy, the part is sent to machine C. If all machines are busy, the part wait for the first one to be free. Simulate for 10 days operation of 8 hours each day.

The program to do the simulation is:

```
        SIMULATE
        GENERATE    12,4.5
        TRANSFER    ALL,AAAA,BBBB,4
AAAA    SEIZE       MACH1
        ADVANCE     14,4.3
        RELEASE     MACH1
        TERMINATE
        SEIZE       MACH2
        ADVANCE     20,5.7
        RELEASE     MACH2
        TERMINATE
BBBB    SEIZE       MACH3
        ADVANCE     30,3.4
        RELEASE     MACH3
        TERMINATE
        GENERATE    480*10
        TERMINATE   1
        START       1
        END
```

Solution

Selected output from running the program is as follows:

```
RELATIVE CLOCK: 4800.0000 ABSOLUTE CLOCK: 4800.0000
```

```
BLOCK   CURRENT   TOTAL   BLOCK   CURRENT   TOTAL
1                   402   BBBB                32
2                   402   12                  32
AAAA                229   13                  32
4             1     229   14                  32
5                   228   15                   1
6                   228   16                   1
7                   141
8             1     141
9                   140
10                  140

        --AVG-UTIL-DURING--

FACILITY   TOTAL   ENTRIES   AVERAGE
                             TIME/XACT
MACH1      0.661     229      13.852
MACH2      0.592     141      20.145
MACH3      0.199      32      29.856
```

As can be seen there were 402 failed parts sent to the machines in the 10 days. 229 were sent to machine A which was busy 66.1% of the time. Machine B repaired 141 parts and was busy 59.2% of the time; machine C received only 32 parts and was busy only 19.9% of the time.

3. THE TRANSFER FUNCTION MODE

The TRANSFER PICK transfers the transactions to different blocks with equal probability. Sometimes you will want to transfer a transaction to a particular set of blocks only certain percentages each. The TRANSFER block in function mode is used for this. There can be several forms but the one mostly used is:

```
TRANSFER  ,FN(name)
```

The function referenced can have blocks in the number pairs in the function definition. For example,

```
FIRST FUNCTION   RN1,D4
.1,BLOCKA/.35,BLOCKB/.8,BLOCKC/1,BLOCKD
```

```
      ..............
      ..............
      TRANSFER   ,FN(FIRST)
```

BLOCKA, BLOCKB, BLOCKC and BLOCKD are block labels. The transaction will be transferred to BLOCKA 10% of the time; to BLOCKB 25% of the time; to BLOCKC 45% of the time and to BLOCKD 15% of the time. This is a very useful form of the TRANSFER block.

Example 18.3

Trucks arrive for service at a repair shop. Interarrival times are 28 ± 6 minutes. 10% of the repairs are of a serious nature and a special crew is called in to do the work. These take 200 ± 60 minutes to complete. The other services can be broken into two types, Type A and Type B. Type A service is required by 30% of the trucks and Type B by the remaining 60%. Both of these are done by the same crew. Type A service takes 45 ± 15 minutes and Type B takes 20 ± 9 minutes. Simulate for 20 shifts of 8 hours each.

Solution

The program to do the simulation is as follows.

```
      SIMULATE
WHICH FUNCTION   RN1,D3
.1,SERVA/.4,SERVB/1,SERVC
      GENERATE   28,6
      TRANSFER   ,FN(WHICH)
SERVA QUEUE      WAIT
      SEIZE      OTHER
      DEPART     WAIT
      ADVANCE    200,60
      RELEASE    OTHER
      TERMINATE
SERVB QUEUE      WAIT
      SEIZE      FIRST
      DEPART     WAIT
      ADVANCE    45,15
      RELEASE    FIRST
      TERMINATE
SERVC QUEUE      WAIT
```

```
      SEIZE      FIRST
      DEPART     WAIT
      ADVANCE    20,6
      RELEASE    FIRST
      TERMINATE
      GENERATE   480*20
      TERMINATE  1
      START      1
      END
```

Selected portions from the output are as follows:

```
RELATIVE CLOCK: 9600.0000 ABSOLUTE CLOCK: 9600.0000

BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL  BLOCK  CURRENT  TOTAL
1                344  11               111  21                   1
2                344  12               111  22                   1
SERVA             31  13               111
4                 31  14               111
5                 31  SERVC         1  201
6                 31  16               200
7                 31  17               200
8                 31  18            1  200
SERVB        1    112  19               199
10               111  20               199

        --AVG-UTIL-DURING--

FACILITY   TOTAL   ENTRIES   AVERAGE
           TIME              TIME/XACT
OTHER      0.609     31       188.703
FIRST      0.949    311        29.303

QUEUE   MAXIMUM   AVERAGE   TOTAL    ZERO     AVERAGE    $AVERAGE
        CONTENTS  CONTENTS  ENTRIES  ENTRIES  TIME/UNIT  TIME/UNIT
WAIT       6       2.327     344       58       64.934     78.102
```

When using the TRANSFER function mode, it is possible to have a function that returns a number. This number refers to the program block number to be transferred to. Recall that the

GPSS processor numbers all of the blocks consecutively during compiling. These numbers are used to determine where the transaction is transferred to. For example, if the block

```
TRANSFER  ,FN(WHERE)
```

returns the value 12, the transaction is routed to the block in numerical position 12. The problem with using this form of the TRANSFER block is that, if any changes are made to the program such as adding blocks, all such TRANSFER blocks must also be changed. It is also possible to have the following:

```
TRANSFER  ,FN(THIRD)+5
```

The function THIRD is referenced and a value is returned, say, 14. Then 5 is added to this and the transaction is routed to the block in position 19.

4. THE TRANSFER PARAMETER MODE

It is possible to transfer to a block number whose value is given by one of the transaction's parameters. The form of this is,

```
TRANSFER  ,PH4
```

In this case, the transaction is routed to the block given by the value of the transaction's fourth parameter. If this value is 15, the transaction is routed to block 15, if the value is 20, the transaction goes to 20, etc.

There is also a provision to have a C operand in this form such as:

```
TRANSFER  ,PH7,3
```

Now, 3 is added to the value stored in the transaction's seventh parameter and the transaction is routed to the block given by this total. For example, If the value stored in parameter 7 was 30, the transaction would be routed to block 33.

The above could also have been coded as:

```
TRANSFER  ,PH7+1
```

The previous exercise could have been written using this form of the TRANSFER block as follows:

```
      SIMULATE
WHICH FUNCTION   RN1,D3
.1,4/.4,10/1,16
      GENERATE   28,6
      ASSIGN     5,FN(WHICH),PH
      TRANSFER   ,PH5
      QUEUE      WAIT
      SEIZE      OTHER
      DEPART     WAIT
      ADVANCE    200,60
      RELEASE    OTHER
      TERMINATE
      QUEUE      WAIT
      SEIZE      FIRST
      DEPART     WAIT
      ADVANCE    45,15
      RELEASE    FIRST
      TERMINATE
      QUEUE      WAIT
      SEIZE      FIRST
      DEPART     WAIT
      ADVANCE    20,6
      RELEASE    FIRST
      TERMINATE
      GENERATE   480*20
      TERMINATE  1
      START      1
      END
```

The output from the above program is identical to the previous one. Because it is possible to have arithmetic in operands, the use of the C operand in the TRANSFER blocks is hardly ever used.

5. THE SUBROUTINE MODE

It is possible to transfer to a subroutine and then return to the main program. This might be done when there are a series of identical blocks that have to be repeatedly referenced. The general form of this is:

```
TRANSFER    SBR,MYSUB,PH2
```

The word SBR must be in operand position A. MYSUB is the block label to transfer to. Upon entering the TRANSFER SUB block, the transaction is unconditionally routed to this block. The block number of the TRANSFER SBR block in placed in the transaction's parameter as specified by the third operand, in this case PH2. Unlike other programming languages, when the transaction finishes with the subroutine, it does not automatically return to the main program. It must be directed back to the block it came from in order to continue moving from block to block. To accomplish this, the block location is placed in the transaction parameter given by the third operand, in the example here, this would be parameter 2. Once the transaction finishes the subroutine it has to be transferred back to the main with the following:

```
TRANSFER    ,PH2+1
```

The addition of 1 to PH2 is essential in order that the transaction returns to the block following the initial subroutine call. Otherwise, the transaction will be transferred back to the original TRANSFER SBR block and an infinite loop would be formed.

6. THE TRANSFER SIMULTANEOUS MODE

Another form of the TRANSFER block is the TRANSFER SIM mode. This was used more in earlier versions of GPSS when it was more important to write code that took the minimum of time for execution. This form of the TRANSFER block makes use of the fact that every time a transaction is delayed it has a switch called the SIM indicator that is put in a set position. Whenever a transaction leaves an ADVANCE block this switch is reset. Also, when a transaction leaves the TRANSFER SIM block, the switch is also reset. The general form of this is:

```
TRANSFER    SIM,LABEL,BACK1
```

The work SIM must be in operand position A, Operands position B is the label of the block where the transaction is routed to when the switch is in a reset position. It is rarely used because the next sequential block is the often one to route the transaction to. IN this regard, it is similar to the TRANSFER BOTH block. Operand C is the block where the transaction is routed to when the switch is in a set position. This is usually at beginning of as series of GATE blocks or other blocks used in logic test.

Consider the following series of blocks which might have to do with a ship ready to enter a harbor. In order for it to do so, three conditions must be satisfied simultaneously:

1. There must be a high tide
2. The single tug boat must available

3. A berth must be free

```
BACK1 GATE       LS          ONE IS THERE A HIGH TIDE?
      GATE       FNU         TUG IS THE TUG AVAILABLE?
      GATE       SNF         BERTH IS A BERTH FREE
      TRANSFER   SIM,,BACK1
```

The first GATE block has to do with the fact that there must be a high tide. The second with the tug and the third with the berth.

Suppose that a transaction enters these blocks. Its SIM switch is reset. If it is not delay at any of the three GATE blocks, it simply leaves the TRANSFER SIM block to the next sequential block. Suppose that there was no high tide. The transaction waits until high tide. Since it has now been delayed, its SIM switch is placed in a set position. If the other two GATE blocks allow it to pass through, it arrives at the TRANSFER SIM block with its SIM indicator in a set position. Thus, it is routed to the block BACK1 as given by the C operand. However, its SIM switch is placed in a reset position. Now, it passed through the three GATE blocks and when it again encounters the TRANSFER SIM block, it passes through to the next sequential block.

Suppose that another transaction enters the series of three blocks at a later time. The tide is low but the tug is busy. The transaction is held at the second gate block until a tug is free. Now, suppose that while the ship was waiting for the tug, the tide changes to a low tide. When the transaction reaches the TRANSFER SIM block the switch is in a set position because of the delay in waiting for the tug boat. Thus, it will be routed to the first GATE block. But now the tide is so the transaction will be delayed here until the time becomes high. The transaction will continue to cycle through the three GATE blocks and the TRANSFER SIM block until all three conditions as given by the GATE blocks are satisfied.

The above could have been done using a single TEST block and Boolean logic (to be covered later) but this causes more execution time than using the three GATE blocks. Whenever possible the analyst is encouraged to avoid the use of the TEST block due to the increase of execution time.

 **Return on CONTENS**

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

Chapter 19
AMPERVARIABLES, DO LOOPS, the PUTPIC, PUTSTRING and GETLIST statements, IF, GOTO and LET statements

AMPERVARIABLES

It is possible to have variables that change in a GPSS/H program each time it is run. We have done this already by re-defining the block to be changed. For example, a program which was run once with 4 workers in a factory had:

```
WORKERS GENERATE   ,,,4
```

After the first run, we might have had statements such as:

```
        START      1
        CLEAR
        RMULT      777
WORKERS GENERATE ,,,5
        START      1
```

Now the program is run a second time but with 5 worker transactions being used in the simulation. If it is desired to run the program again but now with 6 workers, it is easy to add the necessary lines of code. It is possible in GPSS/H to simplify this further by using the concept of ampervariables. These are variables that have their values changed in the program. They are defined by the use of the ampersand as their first character (hence, the name, ampervariables).

GPSS/H allows for 5 types of these ampervariables: integer, real or floating point, 2 character types and external. Integer ampervariables are whole numbers; real or floating point ampervariables are with decimals, character ampervariables are for character strings and external ampervariables refer to external functions and subroutines. In the following discussion only integer and real ampervariables will be used as they are the ones most commonly used.

All ampervariables must be defined prior to their use. They are defined by the statement:

```
        INTEGER     list of ampervariables
```

or

```
REAL        list of ampervariables
```

They may be up to 8 alphanumeric characters in length. Thus,

```
a) INTEGER &I,&JOE,&K123456,&JJJ,&XYZ
b) REAL    &ZX,&KLMN,&TRUCKS,&SPEED
```

a) would define integer ampervariables I, JOE K123456 and JJJ and XYZ. b) would define real ampervariables, ZX, KLMN, TRUCKS and SPEED. In the main program blocks it is then possible to have the following:

```
QUEUE       &I
ADVANCE     &SPEED
GENERATE    ,,,&JJJ
SEIZE       &JOE
ASSIGN      1,&ZX,PL
etc.
```

Integer ampervariables are most commonly used in connection with the GPSS/H DO LOOP, which is covered next.

THE GPSS/H DO LOOP

GPSS/H has DO LOOPS which can be used to greatly shorten the code for control statements. The form is quite similar to that found in other programming languages and is:

```
DO    integer ampervariable=lower limit,upper limit,increment
.....
.....
ENDDO
```

The increment is optional. Here is how the DO LOOP works. The integer ampervariable is set equal to its lower limit and the statements from the DO statement down to the ENDDO are executed. The integer ampervariable is then incremented by the increment. If the increment is missing (it often is), the value is assumed to be 1 by default. The statements are then executed up to the ENDDO. This is continued until the value of the ampervariable is greater than the upper limit. Thus, the lines of code:

```
      DO          &I=2,10
      CLEAR
      RMULT       9
WORKS GENERATE    ,,,&I
      START       1
      ENDDO
```

Would run the program first with I = 2 and then with I = 3, etc. up to and including I = 10 for a total of 9 times. There would be 9 reports written. It is not possible to decrement the ampervariable in a DO LOOP. The lines of code:

```
      DO          &K=4,10,2
      CLEAR
      RMULT       777
      INITIAL     XH(VALUE),&K+3
      START       1
      ENDDO
```

Would run the program first for values of K = 4 and VALUE = 7. The second time, the values would be K = 6 (increment is 2) and XH(VALUE) = 9, etc. up to K = 10 and VALUE = 13.

It is possible to have nested DO loops. If so, each must have its own ENDDO statement:

```
      DO          &J=2,6
      CLEAR
      RMULT       54321
TRUCKS GENERATE   ,,,&J
      DO          &I=1,3
      STORAGE     S(TUGS),&I
      START       1
      ENDDO
      ENDDO
```

The value of &J is first set equal to 2. The block

```
TRUCKS GENERATE  ,,,&J
```

would be

```
TRUCKS GENERATE  ,,,2
```

next, the value of I is 1 and the STORAGE is:

```
STORAGE    S(TUGS),1
```

The program is run for these values. The value of I is next incremented to 2 and the program run with the STORAGE of TUGS equal to 2 (&J remains equal to 2). Thus, the main program will be executed 18 times. (&J = 2,3,4,5,6 and &I = 1,2,3).

The PUTPIC Block

Up to this point, all of our output has been given by the GPSS/H processor. Most of it has been disregarded. This is usually the case as only a few of the many SNA's produced are of interest depending on the particular problem. It is possible to have the processor print out only selected statistics. This is done via the PUTPIC block. One form of it is:

```
PUTPIC  LINES=n,FILE=SYSPRINT,(list of SNA's)
```

The LINES = n indicate how many lines are to be included in the output as a result of the PUTPIC (PUT a PICture on the screen) statement.

The FILES=SYSPRINT puts the output into the report. If this is omitted, the output is written immediately on the screen. This may be all right for some problems but, in most cases, the output will scroll across the screen too rapidly.

It is possible to have any other file listed where the output is to be directed. For example,

```
PUTPIC   LINES=7,FILE=MYFILE,(list of SNA's)
```

would direct the result of the PUTPIC statement to a file named MYFILE. Notice that no extension is used.

The list of SNA's are included in the parenthesis and separated by commas. The parenthesis is optional. The SNA's will be printed out as a result of the PUTPIC statement.

The format where the SNA's are to be printed is specified by using asterisks "*'s" in the lines that follow the PUTPIC statement. These can have decimals in them. For example, if the PUTPIC statement were:

```
      PUTPIC   LINES=3,FILE=SYSPRINT,(&I)
The results for iteration #***
Are given here.
================================
```

If the value of &I was 3, the output would be:

```
The results for iteration # 3
Are given here.
================================
```

In order to add blank lines to the output a 0 (zero) is placed in position 1 of an output line, this will cause a blank line to be printed. The 0 is not printed. Thus,

```
    PUTPIC   LINES=5,FILE=SYSPRINT,(&I,FC(MACH),FR(MACH)/1000)
RESULTS OF SIMULATION STUDY
SIMULATION # ***
THE MACHINE MADE **** PARTS
IT WAS BUSY .***% OF THE TIME
============================
```

Might result in output as follows:

```
RESULTS OF SIMULATION STUDY
SIMULATION # 5
THE MACHINE MADE 354  PARTS
IT WAS BUSY 0.876% OF THE TIME
==============================
```

Notice that arithmetic was used in the output specification of the PUTPIC statement. Use of ampervariables and DO LOOPS can greatly reduce the lines of code used in writing control statements to run programs multiple times. Use of the PUTPIC statement can enhance the output of GPSS/H programs. These features are not found in other versions of GPSS.

THE PUTSTRING STATEMENT

If you only want to have text created either on the screen or in a file, the PUTSTRING statement is used. This is quite simple to use as it serves to place text on the screen as the programmer wants it. This is especially useful for running programs in an interactive mode. The

general form of it is:

```
(label) PUTSTRING  FILE=filename,(' text to be printed ')
```

The filename is the file where the output is to be directed. This is often omitted and the result of the PUTSTRING is sent to the user's input device. Some examples of the PUTSTRING are:

```
a) PUTSTRING (' HELLO THERE ')
b) PUTSTRING (' THIS IS A GPSS/H PROGRAM ')
c) PUTSTRING (' ')
```

The results of a) and b) are to put the messages HELLO THERE and THIS IS A GPSS/H PROGRAM on the screen. The effect of c) is to put a blank line on the screen. (note: you need at least 2 spaces between the apostrophes to obtain a blank line).

You may include standard ASCII alphanumeric code in a PUTSTRING specification. There are typed by holding the <alt> key down on the keyboard and inputting the ASCII numerical specification for the character using the numbers on the numeric key pad, not the ones at the top of the keyboard. For example, the code for the **?** is 219. The PUTSTRING:

```
PUTSTRING  (' ? ? ? IMPORTANT INFORMATION FOLLOWS ? ? ? ')
```

would result in the text

```
? ? ? IMPORTANT INFORMATION FOLLOWS ? ? ?
```

By using the ASCII code for the **?**, which is 220, together with the **?**, it is possible to place the title to a program in a box. For example, you might have something like the following:

```
PUTSTRING  (' ???????????????????? ')
PUTSTRING  (' ?                    ? ')
PUTSTRING  (' ?  SIMULATION OF     ? ')
PUTSTRING  (' ?  A SHOP SYSTEM     ? ')
PUTSTRING  (' ?                    ? ')
PUTSTRING  (' ???????????????????? ')
```

Consult any manual on DOS for a complete listing of the ASCII codes available.

Example 19.1

The owner of a garage stocks snow tires for sale during the winter months. He places one order at the end of summer and cannot receive any more tires if the demand is greater than the supply. Tires cost $20 + $25*(number of tires ordered). The $20 is a fixed cost no matter how many tires are ordered. The tires sell for $45 each. If any tires are left over, there is a penalty of $5 per tire in "holding" costs. If a person wants to buy a tire and it is no longer in stock, the $20 profit is considered as a loss. From past records, the owner feels that the demand for tires is given by the following probability distribution:

Table 19.1 Distribution of demand for tires.

demand	prob.
100	.03
105	.05
110	.10
115	.15
120	.18
125	.14
130	.12
135	.10
140	.08
145	.03
150	.02

Determine the amount of tires the garage owner should order to maximize his expected profit. Simulate for 200 winters.

Solution

The way to do the simulation is to first assume a supply amount of a reasonable amount of tires. Suppose this amount is 120. Then using the data in Table 19.1, a demand is simulated by means of Monte Carlo simulation. Suppose this is 100. This means that the store made a profit of:

100*$45 - $20 - $120*25 - $5*(120 - 100) = $1,380.

If the demand had been 130, the profit would have been:

120*45 - $20 - $120*25 - $20*(130 - 120) = $2,180.

This is done for a large number of possible demands, say 200. The expected profit is then the average of the simulate ones. This is then taken to be the expected profit (or loss) for the given supply amount. The program assumes supply amounts of from 90 to 150 in increments of 5, i. e., amounts of 90, 95, 100, ...,150.

GPSS is ideal for such inventory problems. The program to do the simulation is given below:

```
      SIMULATE
      INTEGER    &I
      PUTSTRING  (' ')
      PUTSTRING  (' ')
      PUTSTRING  (' SIMULATION OF A TIRE SUPPLY PROBLEM')
      PUTSTRING  (' ')
      PUTSTRING  (' A DEALER PLACES AN ORDER FOR TIRES ONLY ONCE')
      PUTSTRING  (' THESE COST $20 + $25*(NUMBER ORDERED) ')
      PUTSTRING  (' TIRES SELL FOR $45 EACH')
      PUTSTRING  (' IF ANY TIRES ARE LEFT OVER, A PENALTY ')
      PUTSTRING  (' OF $5.00 PER TIRE RESULTS')
      PUTSTRING  (' IF A PERSON WANTS TO BUY A TIRE AND IT IS NOT')
      PUTSTRING  (' AVAILABLE, THE $20 PROFIT IS CONSIDERED AS A
LOSS')
      PUTSTRING  (' THE DEMAND PROBABILITY DISTRIBUTION IS GIVEN')
      PUTSTRING  (' ')
      PUTSTRING  (' STAND BACK WHILE THE SIMULATION IS PERFORMED...')
DEMAND FUNCTION  RN1,D11
.03,100/.08,105/.18,110/.33,115/.51,120/.65,125/.77,130
.87,135/.95,140/.98,145/1,150
      GENERATE   ,,,1
BACK1 ASSIGN     1,FN(DEMAND),PH          DETERMINE DEMAND
      TEST GE    XH(STOCK),PH1,DOWN1      ENOUGH SUPPLIED?
      SAVEVALUE  PROFIT+,PH1*45.-(20.+25.*XH(STOCK))-_
                 5.*(XH(STOCK)-PH1),XL
      ADVANCE    1 ONE YEAR PASSES
      TRANSFER   ,BACK1                   BACK FOR ANOTHER YEAR
DOWN1 SAVEVALUE  PROFIT+,XH(STOCK)*45.-(20.+25.*XH(STOCK))-_
                 20.*(PH1-XH(STOCK)),XL
      ADVANCE    1                        ONE YEAR PASSES
```

```
    TRANSFER   ,BACK1                    BACK FOR ANOTHER YEAR
    DO         &I=90,150,5               BEGIN DO LOOP
    CLEAR                                CLEAR FOR NEXT RUN
    RMULT      77777                     RESET RANDOM NUMBERS
    INITIAL    XH(STOCK),&I              INITIALIZE TIRES
    GENERATE   200                       200 YEARS
    SAVEVALUE  RESULT,XL(PROFIT)/200,XL  DETERMINE PROFIT
    TERMINATE  1                         END OF SIMULATION
    START      1,NP
    PUTPIC     LINES=5,FILE=SYSPRINT,(&I,XL(RESULT))
|=================================================|
|     RESULTS OF SIMULATION FOR TIRE SHOP         |
|     NUMBER OF ITEMS IN STOCK WAS          ***   |
|     WITH THESE, THE EXPECTED PROFIT WAS  ****.** |
|=================================================|
    ENDDO
    END
```

Now when the program is executed, the screen will have the following on it:

```
   SIMULATION OF A TIRE SUPPLY PROBLEM
   A DEALER PLACES AN ORDER FOR TIRES ONLY ONCE
   THESE COST $20 + $25*(NUMBER ORDERED)
   TIRES SELL FOR $45 EACH')
   IF ANY TIRES ARE LEFT OVER, A PENALTY
   OF $5.00 PER TIRE RESULTS
   IF A PERSON WANTS TO BUY A TIRE AND IT IS NOT
   AVAILABLE, THE $20 PROFIT IS CONSIDERED AS A LOSS
   THE DEMAND PROBABILITY DISTRIBUTION IS GIVEN
   STAND BACK WHILE THE SIMULATION IS PERFORMED...
```

The output file will be limited to that given by the listing of the program and then by the PUTPIC statement. The output from this PUTPIC statement will be as follows:

```
|=================================================|
|     RESULTS OF SIMULATION FOR TIRE SHOP         |
|     NUMBER OF ITEMS IN STOCK WAS          90    |
|     WITH THESE, THE EXPECTED PROFIT WAS  1105.50|
|=================================================|
```

```
|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS          95   |
|      WITH THESE, THE EXPECTED PROFIT WAS  1305.50 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS         100   |
|      WITH THESE, THE EXPECTED PROFIT WAS  1505.50 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS         105   |
|      WITH THESE, THE EXPECTED PROFIT WAS  1691.50 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS         110   |
|      WITH THESE, THE EXPECTED PROFIT WAS  1874.00 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS         115   |
|      WITH THESE, THE EXPECTED PROFIT WAS  2023.25 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
|      NUMBER OF ITEMS IN STOCK WAS         120   |
|      WITH THESE, THE EXPECTED PROFIT WAS  2121.75 |
|=================================================|

|=================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP        |
```

```
|      NUMBER OF ITEMS IN STOCK WAS          125    |
|      WITH THESE, THE EXPECTED PROFIT WAS   2145.00 |
|====================================================|

|====================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP           |
|      NUMBER OF ITEMS IN STOCK WAS          130     |
|      WITH THESE, THE EXPECTED PROFIT WAS   2121.00 |
|====================================================|

|====================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP           |
|      NUMBER OF ITEMS IN STOCK WAS          135     |
|      WITH THESE, THE EXPECTED PROFIT WAS   2053.25 |
|====================================================|

|====================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP           |
|      NUMBER OF ITEMS IN STOCK WAS          140     |
|      WITH THESE, THE EXPECTED PROFIT WAS   1943.50 |
|====================================================|

|====================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP           |
|      NUMBER OF ITEMS IN STOCK WAS          145     |
|      WITH THESE, THE EXPECTED PROFIT WAS   1809.25 |
|====================================================|

|====================================================|
|      RESULTS OF SIMULATION FOR TIRE SHOP           |
|      NUMBER OF ITEMS IN STOCK WAS          150     |
|      WITH THESE, THE EXPECTED PROFIT WAS   1666.25 |
|====================================================|
```

From the output it is found that the optimum number of tires to order is 125 which will result in an expected profit of $2,121.45. However, if 120 or 130 are ordered, the expected profit is only slightly less. If desired, the program can be re-run with the supply amounts taken as being from 120 to 130 in increments of 1.

As can be seen, the output is greatly reduced from what is normally obtained.

note: This example is modified from one found in "Introduction to Operations Research Models", by Cooper, Bhat and LeBlanc, W. B. Saunders Co, Philadelphia, 1977, page 344.

The GETLIST STATEMENT

Once ampervariables are defined they can be read into the program by means of the GETLIST statement. This causes the program execution to stop until the ampervariables as specified in the GETLIST specification are read from the user's input device. A prompt appears on the screen until the data in input. The general form of this is:

```
label GETLIST (FILE=filename),(list of ampervariables)
```

The label is optional. For FILE=filename the common one is GUSER which is used for interactive screen input. If more than one ampervariable is read it by a single GETLIST statement, they must be separated by blanks not commas. For example,

```
GETLIST FILE=GUSER,&I,&J,&K
```

If values of 2, 3, and 7 are to be assigned to &I, &J and &K respectively, the data must be input as:

```
2 3 7
```

If a GETLIST statement asks for more than 1 value to be read in and less than this number is input, the prompt remains on the screen until all the data is input. The FILE=GUSER specification can be omitted as GPSS/H assumes this is the file by default.

Example 19.2

Go back to Example 19.1 and add the necessary code so that the number of years to simulate for is a variable. Also, have the supply amounts and the increments as variables.

Solution

The changes are as follows:

Replace INTEGER &I with INTEGER &I,&J,&K,&L,&M

Add the code:

```
PUTSTRING (' HOW MANY YEARS DO YOU WANT TO SIMULATE FOR')
GETLIST   FILE=GUSER,&J
PUTSTRING (' ')
PUTSTRING (' NEXT WE NEED THE SUPPLY AMOUNTS TO CONSIDER IN')
PUTSTRING (' THE SIMULATION. INPUT THE BEGINNING AMOUNT,')
PUTSTRING (' THE END AMOUNT AND THE INCREMENT SEPARATED BY')
PUTSTRING (' BLANKS - NOT COMMAS!!')
PUTSTRING (' ')
GETLIST   FILE=GUSER,&K,&L,&M
```

Replace DO &I=90,150,5 with DO &I=&K,&L,&M

```
GENERATE  200 whit GENERATE &J
```

and change

```
SAVEVALUE RESULT,XL(PROFIT)/200,XL
```

to

```
SAVEVALUE RESULT,XL(PROFIT)/&J,XL
```

The IF and GOTO and HERE Statements in GPSS/H

It is possible to have IF statements in GPSS/H. These can be used in the control statements to check on the input data or after the program has executed to prompt the user to re-run the program, often with different data. The form of the IF statement is:

```
label IF
      (condition)
      ..........
      ..........
      ENDIF
```

If the condition is true, the group of statements after it are executed. These conditions are logical statements such as

```
      &I+&J<=&K
      &KLM-&MMM=0
      &YES'E'1
```

It is possible to have ELSEIF and ELSE statements such as are found in other programming languages. Their form would be:

```
label IF condition 1
      ....
      ....
      ELSEIF condition 2
      ....
      ....
      ELSE condition 3
      ....
      ....
      ENDIF
```

An example of these might be:

```
      IF &I'E'2
      LET &J=0
      ELSEIF &I'E'3
      LET &J=1
      ELSE
      LET &J=2
      ENDIF
```

Often the GOTO is used with the IF statement. This is simply

```
      GOTO label
```

For example, you might have:

```
      PUTSTRING (' THE DATA YOU TYPED IN IS AS FOLLOWS')
      PUTPIC    LINES=3,&I,&J,&SPEED
      THE VALUE OF I = ***
      THE VALUE OF J = ***
```

```
THE SPEED IS = ***.**
PUTSTRING (' ')
PUTSTRING (' ARE YOU HAPPY WITH THESE?')
PUTSTRING (' RESPOND: 1 = YES; 0 = NO')
GETLIST FILE=GUSER,&L
IF &L'E'0
GOTO TRYAGAIN
ENDIF
```

The effect of the above is to prompt the user to examine the data already input. If the data is not satisfactory, control returns to where the data was originally input.

It is possible to have control return to a target of the GOTO that is a dummy statement known at the HERE statement. The form of this is:

```
label HERE
```

Thus, in the previous example, there might have been a statement:

```
TRYAGAIN HERE
```

This statement is analogous to the Fortran CONTINUE statement.

The LET Statement

In the discussion above, ampervariables were initialized when they were read into the program. It is also possible to initialize them by means of the LET statement. The form is quite simple. It is simply:

```
label LET ampervariable=value
```

For example,

```
LET &I=12
LET &XONE=&SPEED/360.5
LET &AREA=XL(LENGTH)*XL(WIDTH)
```

It is possible to have a LET statement as the target of a GOTO statement. **2**

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul **Mine Design Using Simulation**

CHAPTER 20.
The SELECT and COUNT blocks

GPSS has several powerful blocks that can be used to search several entities and test their attributes for a specific condition when a transaction enter the block. When this condition is met, a record of this is placed in one of the transaction's parameters. Such scans are common in real life situations. For example, a person entering a bank that has individual queues at each teller will scan these queues, determine which is the shortest and then join the queue that is the shortest. The block to do this scan and test at the same time is the SELECT block. When a transaction enters this block, a scan of selected entity members is made. When one of these scanned members is found that satisfies some stated condition, the scan is terminated. Examples of where such a block might be used to model real life situations are as follows:

1. A person enters a bank that has 5 tellers each with a queue of people waiting for service. The person will look at each queue and determine which is the shortest and then join that one.
2. A part comes along an assembly line where 3 machines can work on it. The part will be sent to the first machine that is not in use. If all three machines are in use, the part is sent to another section of the plant.

In order to do this scanning and testing the SELECT block needs to know what entities to scan, what the test is, where to put the result of the scan and what to do if the scan is not successful. As a result, the SELECT block can have up to 6 operands. As such, it is the most complicated block since the GENERATE block. One general form of the SELECT block is:

```
SELECT R A,B,C,D,E,(F)
```

R is a relational operator which is one of the following:

G	greater than
GE	greater than or equal to
E	equal to
NE	not equal to
LE	less than or equal
L	less than

A is the parameter number into which the first entity number which satisfies the test is to be placed. Thus, if queues numbered from 5 to 8 are scanned and queue number 7 satisfies the condition, the number 7 is placed in the transaction's parameter given by A.

B is the smallest number of the entities to be scanned.

C is the largest number of the entities to be scanned.

E the family name of the SNA to be used in the scan. This might be F, PH, S, Q, etc.

F (optional) This is a block label where the transaction is to be transferred to if the scan is not successful.

The SELECT block is best understood by considering examples of its use.

a) SELECT E 3PF,1,5,2,Q

a) When a transaction enters this block, a scan will be made of the queues from 1 to 5. These will be tested to see if any has a queue length of 2. If so, the number of the queue will be placed in the transaction's parameter #3. If no queue length of ones from 1 to 5 has a queue length of 2, the transaction moves to the next sequential block. If the queue lengths at all the queues is zero except at queues 4 and 5 where they are both 3, the number 4 is placed in the transaction's full word parameter number 3.

b) SELECT G 5PH,3,7,250,FR

b) A scan is made of facilities numbered 3 to 7 starting with facility 3 and going up to facility 7. Once one is found to have a fractional utilization greater than .250 (recall that the fractional utilization is expressed in parts per thousand), the scan is stopped and the facility number is placed in the transaction's halfword parameter number 5. If no facility from 3 to 7 has a fractional utilization greater than .250, the transaction moves to the next sequential block.

c) SELECT LE 10PH,1,3,1,R,AWAY

c) Storages numbered from 1 to 3 are scanned. If one has a remaining storage of less than or equal to 1, the number of it is copied in the transaction's 10th parameter. If no storage in the scan satisfies the criteria, the transaction is routed to the block with the label AWAY.

d) SELECT NE 4PH,7,12,0,PH

d) The scan is of the transaction's half word parameters numbered from 7 to 12. Once one of these is found to be not equal to zero, the number of it is copied in the transaction's 4th parameter.

e) SELECT E FN(ONE),PH3,PH4,3,Q,DOWN

e) The parameter to place the result of the scan (if successful) is given by reference to the function ONE. Suppose it is 2. The scan will be to see if any of the queue scanned have a length of 3. The queues to be scanned will depend on the values of the transaction's 3rd and 4th parameters. Suppose these are 5 and 9 respectively. Then queues 5 through 9 are scanned and, if any has a length of 3, the number of it is placed in the transaction's parameter number 2. If none of the queues is equal to 3, the transaction is routed to the block with the label DOWN.

Example 20.1

Parts come along an assembly line to be worked on by one of three identical machines. The parts arrive at a rate of one every 8 ± 4.5 minutes. The machines finish a part in 20 +/- 6 minutes. If a machine is free, the part is worked on by that machine. But if all three machines are busy, the part is sent away to another part of the factory. Determine the utilization of the 3 machines and how many parts are sent away for 20 shifts of 480 minutes each.

Solution

The program to do the simulation is given below:

```
      SIMULATE
TIMES GENERATE   8,4.5              PARTS ARRIVE
      SELECT E   2,1,3,0,F,AWAY     IS A MACHINE BUSY?
      SEIZE      PH2                USE THE FREE MACHINE
      ADVANCE    20,5               PART WORKED ON
      RELEASE    PH2                FREE THE MACHINE
      TERMINATE                     PART DONE
AWAY  TERMINATE                     FOR WHEN MACHINES ALL BUSY
      GENERATE   480*20             20 SHIFTS
      TERMINATE  1                  END OF SIMULATION
      START      1,NP
      PUTPIC     LINES=7,FILE=SYSPRINT,(N(TIMES),N(AWAY))_
      ,(FR1,FR2,FR3)
0================================================
0 THE NUMBER OF PARTS ENTERING THE SYSTEM = ****   =
0 THE NUMBER SENT TO OTHER MACHINES       = ***    =
0 THE UTILIZATION OF THE 1ST MACHINE WAS  = ***    =
0 THE UTILIZATION OF THE 2ND MACHINE WAS  = ***    =
```

```
0 THE UTILIZATION OF THE 3RD MACHINE WAS  = ***    =
0================================================
END
```

The output from the program is as follows:

```
================================================
 THE NUMBER OF PARTS ENTERING THE SYSTEM = 1198   =
 THE NUMBER SENT TO OTHER MACHINES       = 151    =
 THE UTILIZATION OF THE 1ST MACHINE WAS  = 816    =
 THE UTILIZATION OF THE 2ND MACHINE WAS  = 751    =
 THE UTILIZATION OF THE 3RD MACHINE WAS  = 625    =
================================================
```

For the 20 shifts 1198 parts came to the 3 machines. 151 were turned away because all three machines were busy. The utilization of the three machines was .816 for machine 1, 751 for machine 2 and .625 for machine 3. The reason for the decrease in utilization is because when a new part arrives at the machines, the scan is to see is any of the three machines is free. The scan stops when one is found to be free and the scanning always starts at the first machine.

This problem could also have been solved using the TRANSFER ALL block.

Notice that in this example the parameter type was not specified. As long as you only have half word parameters, this is all right. But if the transaction had different type of parameters, it is necessary to specify the type in the SELECT block. For example,

```
GENERATE 20,6.5,,,,2PF,10PF
........ ..................
........ ..................
SELECT E 2,1,5,1,Q
```

would give an error. The correct form of the SELECT block would have to be:

```
SELECT E 2PH,1,5,1,Q (or 2PF)
```

The COUNT Block

The COUNT block resembles the SELECT block in that when a transaction enters it, it triggers a scan of specified entities. The result of the scan is placed in a specified parameter. In the case of the

SELECT block, once the scan finds an entity to satisfy the given test, the scan is over. The COUNT block counts the number of the entities that satisfy the criteria and placed the number counted that satisfy the test criteria into the transaction's specified parameter. One general form of the COUNT block is:

```
COUNT R  A,B,C,D,E
```

R is one of the relational operators used in the SELECT block. Thus, it must be one of the following: G, GE, E, NE, L, or LE. A, B, C, D, and E have the same meanings are the operands in the SELECT block.

Some examples of this block are:

```
a) COUNT E 1,1,4,0,Q
b) COUNT G 3,3,6,250,FR
```

In a), a count is made of all the queues from 1 to 4 that have lengths 0. This number is placed in the transaction's parameter number 1.

In b), a count is made of the facilities from 3 to 6 which have fractional utilization greater than 250. The number of these facilities is placed in the transaction's parameter number 3.

Since the count of the entities in a COUNT block will always be a number, there cannot be any F operand.

Other forms of the SELECT and COUNT Blocks

1. The SELECT block in MIN/MAX mode.

The SELECT block can be used to scan a group of entities and determine which one has a maximum (or minimum) value. The general form of this is:

```
SELECT MIN {or MAX) A,B,C,,E
```

The word MIN (or MAX) must in the auxiliary operator (aux op) position, which is one position away from the SELECT block. The A, B, C, and E operands are the same as for the regular SELECT block described previously. There is no D operand.

When a transaction enters this form of SELECT block, a scan is made of the entities specified by the B and C operands. The processor selects the minimum (or maximum) from the entity class

and places that number into the transaction's parameter number as specified by the A operand.

```
a) SELECT MIN 3,1,4,,FR
b) SELECT MAX 1,3,7,,Q
c) SELECT MIN 5,1,3,,R
```

In a), facilities from 1 to 4 are scanned and the processor will place the number of the one that has the lease fractional utilization into the transaction's parameter number 3. Thus, if facility 1 had a FR of 350, facility 2 of 599, facility 3 of 500 and facility 4 of 222, the number 4 would be in the transaction's 4th parameter.

In b), queues 3 to 7 are scanned. The number which has the greatest length is placed in the transaction's parameter number 1. In case of a tie, i; e', if both queue 4 and queue 7 had equal lengths of 5 and this was the maximum the number 4 is placed in parameter 1. This is also the case if SELECT MIN is used.

In c), storages from 1 to 3 are scanned and the one with the greatest remaining storage is placed in the transaction's parameter number 1.

2. Use with Logic Switches.

A SELECT or COUNT block can be used with logic switches. The general form is

```
SELECT LS (or LR) A,B,C,,,(F)
```

Operands D and E are omitted and F is optional.

The scan is made of logic switches from B (minimum value) to C (maximum value). As soon as the processor encounters one that satisfies the test (either LS for set or LR for reset), the scan is finished and the number of the logic switch is placed in the transaction's parameter number given by operand A. The F operand is a block label where the transaction is routed to if no logic switch satisfies the scan criteria.

The COUNT block in this mode is

```
COUNT LS (or LR) A,B,C
```

The only difference in form from the SELECT block is that the operand F is not used. In this case, a scan is made of the logic switched from B to C and the number of them in set (or reset) condition is placed in parameter number as given by operand A. For example, suppose a transaction entered the following two sequential blocks in a program.

```
SELECT LS  3,2,6
COUNT  LR  4,2,6
```

and suppose that the logic switcher 2 through 6 were as follows:

```
LS2 is reset
LS3 is reset
LS4 is set
LS5 is reset
LS6 is set
```

The transaction would have a 4 in parameter 3 and a 3 in parameter 4.

3. Use with Facilities and Storages

The form of the SELECT block is

```
SELECT (aux op) A,B,C,,,(F)
```

where (aux op) can be one of the following:

aux op	meaning
U	facility in use
NU	facility not in use
SE	storage empty
SNE	storage not empty
SF	storage full
SNF	storage not full

There are no D or E operands and the F operand is optional. The meaning of the operands is the same as before.

The general form of the COUNT block in this mode is:

```
COUNT (aux op) A,B,C
```

A scan is done of the entities and the number of them satisfying the criteria is places in the transaction's parameter number as specified by the A operand. Several examples of these blocks are:

```
a) SELECT NU 1,3,6
b) COUNT NU 2,3,6
c) SELECT SE 3,1,7,,,AWAY
d) COUNT SF 4,1,7
```

In a), the scan is of facilities 3 to 6. Once one is found to be not in use, the scan is finished and the number of the facility is placed in the transaction's parameter number 1.

In b), a count is made of the facilities from 3 to 6 that are not in use. This number is placed in parameter number 3.

In c), a scan of storages from 1 to 7 is made to determine if any are empty. The number of the first one to satisfy the criteria if placed in parameter 3 and the scan is stopped. If no storage is found to satisfy the criteria, the transaction is sent to the block with the label AWAY.

In d), a count is made of the full storages from 1 to 7. This number is placed in parameter 4.

Example 20.2

Did you ever wonder why banks, post offices, airline ticket agents and other places where multiple servers are used now have customers wait in an individual queue rather than forming separate queues at each teller or agent? The single queue system is known as a "quickline" system. This example will illustrate why a quickline system is better than individual queues.

Suppose customers arrive in a store that has 6 clerks behind desks to server the customers. Customers come is and first see if any clerk is free. If so, the customer will go to that desk. If all the clerks are busy, the person will go to the back of the shortest queue. Once at a desk, no queue jumping is allowed.

Customers arrive in a Poisson stream, with an interrarival time of 10 seconds. A customer will transact business are given in Table 20.1:

Table 20.1 Business data for customers

business	% of time	time taken
type 1	28	(25,4)

type 2	17	(32,5)
type 3	30	(60,10)
type 4	25	(80,10)

The time taken is from the normal distribution so that the figures in the table are really (mean, std. dev.). Model this system with both individual queues and the quickline. The store works 10 hours straight.

Solution

The program to model the quickline is quite easy to write using the clerks as storages. The listing of it is as follows:

```
      SIMULATE
      INTEGER    &I
MEAN  FUNCTION   RN1,D4
.28,25/.4,32/.7,60/1,80
STDDEV FUNCTION  PH1,D4
25,4/32,6/60,10/80,10
      STORAGE    S(TELLER),6
      GENERATE   RVEXPO(1,10)          CUSTOMERS ARRIVE
      ASSIGN     1,FN(MEAN),PH         DETERMINE MEAN TIME FOR BUSINESS
      ASSIGN     2,FN(STDDEV)          DETERMINE STD. DEV. FOR BUSINESS
      QUEUE      QLINE                 FORM SINGLE QUEUE
      ENTER      TELLER                ENGAGE A TELLER
      DEPART     QLINE                 LEAVE THE QUEUE
      ADVANCE    RVNORM(1,PH1,PH2)     DO BUSINESS
      LEAVE      TELLER                FREE THE TELLER
      TERMINATE                        LEAVE THE BANK
      GENERATE   3600*10               10 HOURS PASS
      TERMINATE  1                     END OF DAY
      DO    &I=1,10                    SIMULATE FOR 10 DAYS
      CLEAR
      START      1
      PUTPIC     LINES=7,FILE=SYSPRINT,(&I,QM(QLINE),QX(QLINE),_
                 QA(QLINE),QZ(QLINE))
0=====================================
0 SIMULATION RESULTS FOR DAY   ****    =
```

```
0 MAXIMUM QUEUE WAS                ***     =
0 AVERAGE TIME IN QUEUE WAS     ****.**  =
0 AVERAGE QUEUE LENGTH WAS      ****.**  =
0 NUMBER OF ZERO ENTRIES WAS     ***     =
0=====================================
      ENDDO
      END
```

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul **Mine Design Using Simulation**

CHAPTER 21
MATRICES

GPSS/H allows for the use of matrices in a manner similar to that found in other computer languages. The matrix has to be first defined. This means specifying the number of rows, the number of columns, the type of elements that will be in the matrix and the name (or number) of the matrix. A savevalue can be considered as a linear array. A matrix can be considered as a savevalue with 2 or more dimensions.

The general form of the statement that specifies the matrix is:

```
name (or number) MATRIX type,rows,columns
```

The name (or number) follows the usual rules as for naming savevalues. The type of a matrix is one of the following 4:

a) MX, full word matrix savevalue.
b) MH, half word matrix savevalue.
c) MB, bit word matrix savevalue.
d) ML, floating point matrix savevalue.

In a) and b) the M in the type specification can be omitted. It is necessary in c) and d).

The size and integer nature of the elements used in the various types of matrices is the same as for regular savevalues.

Some examples of matrix definitions are:

```
a) FIRST MH,1,3
b) 4 ML,2,10
c) LAST X,3,3
d) OTHER H,5,4
```

In a), FIRST is half word matrix having 1 row and 3 columns, i. e., it might be as follows:

(2 4 -3)

Here the values of the matrix are (1,1) is 2; (1,2) is 4 and (1,3) is -3.

In b), 4 is a floating point matrix with 2 rows and 10 columns.

In c), LAST is a full word matrix of size 3 by 3.

In d), OTHER is declared to be a half word matrix with 5 rows and 4 columns.

Giving Initial Values to Matrices

Once a matrix is defined via the matrix declaration statement, all its elements are set to zero. It is possible to have initial values assigned to the various elements using the INITIAL statement, just as was done for ordinary savevalues. When initializing ordinary savevalues, two forms were possible. One was the older form which was using the dollar sign "$" and the other was using left and right parentheses, only the $ sign is allowed for matrices. Thus, whereas one can have:

```
INITIAL  X$JOEY,4
INITIAL  X(TOMMY),-5/XH(SAM),10/XL$BILLY),12.345
```

When initializing matrices the only form one can use is with the $ sign. Thus, one might have:

```
FIRST MATRIX   MH,1,3
      INITIAL  MH$FIRST(1,1),2/MH$FIRST(1,2),-4
```

The elements of the 1 by 3 dimensioned half word matrix are:

(1 -4 0)

Example 21.1

The following example is from Banks et. al. (1989). In the manufacture of tee shirts the following steps are taken per dozen shirts along an assembly line.

1. Both long and shirt sleeve tee shirts come along the assembly line at the rate of one dozen every 8 ± 1 minute.

2. They are transported to a staging area in 2 minutes. A single worker prepares a bundle of these in 3.75 ± 1 minutes.

3. At the first work station a single worker closes the first shoulder in 3.5 ± 1 minutes.

4. Two workers then sew the collars in 7 ± 2 minutes.

5. A single worker covers the seams in 3 ± .75 minutes.

6. A single worker closes the second shoulder in 3.25 ± 1 minutes.

7. At the next station 5 workers take 11 ± 3 minutes to set the sleeves.

8. Two inspectors are available. Their inspection times vary as shown in Table 21.1:

Table 21.1 Inspection times for the 2 inspectors.

	short sleeve	long sleeve
inspector 1	6 ± 2	7 ± 2
inspector 2	8 ± 1	9 ± 1

9. 5% of all tee shirts are rejected.

Determine how long it will take for 100 dozen tee shirts are finished.

Solution

The solution will utilize a matrix that contains the mean inspection times as given in Table 21.1. The matrix will be of size 2 by 2 and the elements the means inspection times. If the tee shirt is a short sleeve one, it will have a 1 in parameter 1; otherwise it will have a 2 in parameter 1. The program to do the simulation is:

```
      SIMULATE
SERV  MATRIX     MH,2,2
      INITIAL    MH$SERV(1,1),6/MH$SERV(1,2),7
      INITIAL    MH$SERV(2,1),8/MH$SERV(2,2),9
      STORAGE    S(WRK2),2/S(WRK5),3
      GENERATE   8,1
      ASSIGN     1,1,PH
      TRANSFER   ,DOWN
```

```
      GENERATE   8,1
      ASSIGN     1,2,PH
DOWN  ADVANCE    2
      SEIZE      SERV1
      ADVANCE    3.75,1
      RELEASE    SERV1
      SEIZE      WRK1
      ADVANCE    3.5,1
      RELEASE    WRK1
      ENTER      WRK2
      ADVANCE    7.2
      LEAVE      WRK2
      SEIZE      WRK3
      ADVANCE    3,.75
      RELEASE    WRK3
      SEIZE      WRK4
      ADVANCE    3.25,1
      RELEASE    WRK4
      ENTER      WRK5
      ADVANCE    11,3
      LEAVE      WRK5
      TRANSFER   BOTH,,OTHER
      SEIZE      INSP1
      ADVANCE    MH$SERV(1,PH1),2
      RELEASE    INSP1
      TRANSFER   ,BACK
OTHER SEIZE      INSP2
      ADVANCE    MH$SERV(2,PH1),1
      RELEASE    INSP2
BACK  TRANSFER   .05,,REJ
      TEST E     PH1,1,TYPE2
TYPE1 TERMINATE  1
TYPE2 TERMINATE  1
REJ   TERMINATE
      START      100,NP
      PUTPIC     LINES=14,FILE=SYSPRINT,(N(TYPE1),N(TYPE2),_
                 FR(SERV1)/1000,FR(WRK1)/1000,SR(WRK2)/1000,_
                 FR(WRK3)/1000,FR(WRK4)/1000,SR(WRK5)/1000,_
                 FR(INSP1)/1000,FR(INSP2)/1000,AC1)
```

```
0 |=================================================|
  | NO. DOZENS OF SHORT SLEEVE TEE SHIRTS MADE =  ***      |
  | NO. DOZENS OF LONG SLEEVE TEE SHIRTS MADE  =  ***      |
  | UTILIZATION OF FIRST WORK STATION          =    *.***  |
  | UTILIZATION OF SECOND WORK STATION         =    *.***  |
  | UTILIZATION OF THIRD WORK STATION          =    *.***  |
  | UTILIZATION OF FOURTH WORK STATION         =    *.***  |
  | UTILIZATION OF FIFTH WORK STATION          =    *.***  |
  | UTILIZATION OF SIXTH WORK STATION          =    *.***  |
  | UTILIZATION OF FIRST INSPECTOR             =    *.***  |
  | UTILIZATION OF SECOND INSPECTOR            =    *.***  |
  |                                                        |
  | TIME TO MAKE THE 100 DOZEN TEE SHIRTS WAS  ****.** MIN |
  |=================================================|
     END
```

The output from the program is:

```
|=================================================|
| NO. DOZENS OF SHORT SLEEVE TEE SHIRTS MADE =   47      |
| NO. DOZENS OF LONG SLEEVE TEE SHIRTS MADE  =   53      |
| UTILIZATION OF FIRST WORK STATION          =    0.905  |
| UTILIZATION OF SECOND WORK STATION         =    0.844  |
| UTILIZATION OF THIRD WORK STATION          =    0.861  |
| UTILIZATION OF FOURTH WORK STATION         =    0.733  |
| UTILIZATION OF FIFTH WORK STATION          =    0.770  |
| UTILIZATION OF SIXTH WORK STATION          =    0.830  |
| UTILIZATION OF FIRST INSPECTOR             =    0.807  |
| UTILIZATION OF SECOND INSPECTOR            =    0.823  |
|                                                        |
| TIME TO MAKE THE 100 DOZEN TEE SHIRTS WAS   481.52 MIN |
|=================================================|
```

As can be seen, the system is fairly well balanced. It took around 482 minutes to make the 100 dozen tee shirts (47 sort sleeve and 53 long sleeve). The program should be run several more times with different random numbers to see how much the above results change.

The Matrix Savevalue Block

Matrices can have their elements modified by having a transaction move into a block known as the MSAVEVALUE block. The general form of it is:

```
MSAVEVALUE name(or number),B,C,D,E
```

where B is the row
C is the column
D is the new value
E refers to the matrix type

Some examples of this are:

```
a) MSAVEVALUE JOE,1,2,5,MH
b) MSAVEVALUE 4,4,4,4,MX
c) MSAVEVALUE TOMMY,1,10,5.678,ML
d) MSAVEVALUE BETTY,FN(JOE),PH2,X(BILL),H
```

In a), the element (1,2) of the matrix JOE is given the value 5. JOE is a halfword matrix.

In b), the element (4,4) of matrix 4 is given the value 4. the matrix 4 is a full word matrix.

In c), the element (1,10) of the matrix TOMMY is given the value of 5.678. TOMMY is a floating point matrix.

In d), the matrix BETTY is given a value given by savevalue BILL. This will go in the element referenced by the function JOE and the transaction's 2nd half word parameter. BETTY is a half word matrix.

Matrix savevalues can also be used in increment and decrement mode just as ordinary savevalues. Thus,

```
MSAVEVALUE FIRST+,2,5,8,MH
```

will have the element at (2,5) incremented by 8.

Example (from Schriber)

The following is a rather remarkable example of the use of matrices for greatly shortening the programming lines. It can be solved in a straight forward manner using many different segments but can also be solved by using matrices.

A certain production shop is composed of 6 different groups of machines. Each group consists of a certain number of machines of a given kind, as indicated by Table 21.1:

Table 21.1 Composition of Machine Groups

Group Number	Machines in Kind	Group Number
1	Casting units	14
2	Lathes	5
3	Planers	4
4	Drill presses	8
5	Shapers	16
6	Polishing machines	4

Three different types of jobs move through the production shop. These jobs are designated as Type 1, Type 2, and Type 3. Each job-type requires that operations be performed at specified kinds of machines in a specified sequence. The total number and kind of machines each job-type must visit, and the corresponding visitation sequences, are shown in Table 21.2. For example, jobs of Type 1 must visit a total of four machines. The machines themselves, listed in the sequence in which they must be visited, are casting unit, planer, lather, and polishing machine. The table also shows the mean time required by each job-type for each operation that must be performed on it. For example, the casting unit operation for job-type 1 requires 125 minutes, on average. These operation times are all exponentially distributed. Jobs arrive at the shop in a Poisson stream at a mean rate of 50 jobs per 8 hour day. 24% of the jobs are of Type 1, 44% of Type 2 and the rest of Type 3.

Table 21.2. Visitation Sequence and Times for the Jobs

job type	total number of machine to be visited	machines visitation sequence	mean operation time (min.)

1	4	casting unit planer lathe polisher	125 35 20 60
2	3	shaper drill press lathe	105 90 65
3	5	casting unit shaper drill press planer polisher	235 250 50 30 25

Build a GPSS model which simulates the operation of the productions shop. Run the model for the equivalent of 5 40-hour working weeks.

Solution

It would be a straight forward and simple example to model using different model segment with the job types arriving and then being sent to the appropriate segment depending on which type each was. An approach to the solution using matrices is as follows:

Table 21.3 gives the "visitation sequence" for the job types in matrix form.

Table 21.3. The "Visitation Sequence" Matrix

rows (job types)	columns (number of machine groups yet to be visited)				
	1	2	3	4	5

1 2 3	6 2 3 1 2 4 5 6 3 4 5 1

The above matrix give the visitation sequence for each job type for the machine groups yet to be visited. There are three rows in the matrix, one for each job type. The column entries are the jobs yet to be done for the job types. Thus, for job type 1, the sequence of doing the jobs is 1, 3, 2, and then 6. Refer to Table 21.2 for these numbers. Another matrix is shown in Table 21.4 which gives the corresponding times in seconds for each job type. Again, this matrix gives the times for each job type yet to be done.

Table 21.4 Mean Operation Time Matrix

rows (job types)	columns (number of machine groups yet to be visited)				
	1	2	3	4	5
1	600	200	350	1250	
2	650	900	1050		
3	250	300	500	2500	2350

The program to solve the problem is given next:

```
      SIMULATE
      INTEGER    &I
GROUP FUNCTION   PH1,D3
1,4/2,3/3,5
JTYPE FUNCTION   RN1,D3
.24,1/.68,2/1,3
1     MATRIX     MH,3,5
      INITIAL    MH1(1,1),6/MH1(1,2),2/MH1(1,3),3/MH1(1,4),1
      INITIAL    MH1(2,1),2/MH1(2,2),4/MH1(2,3),5
      INITIAL    MH1(3,1),6/MH1(3,2),3/MH1(3,3),4/MH1(3,4),5
      INITIAL    MH1(3,5),1
2     MATRIX     MH,3,5
```

```
        INITIAL    MH2(1,1),600/MH2(1,2),200/MH2(1,3),350
        INITIAL    MH2(1,4),1250
        INITIAL    MH2(2,1),650/MH2(2,2),900/MH2(2,3),1050
        INITIAL    MH2(3,1),250/MH2(3,2),300/MH2(3,3),500
        INITIAL    MH2(3,4),2500/MH2(3,5),2350
        STORAGE    S1,14/S2,5/S3,4/S4,8/S5,16/S6,4
1       TABLE      M1,2400,2400,10
2       TABLE      M1,2400,2400,10
3       TABLE      M1,2400,2400,10
TJOBS   TABLE      V(NUMB1),10,10,5
NUMB1   VARIABLE   W(AAA)+W(BBB)+W(CCC)
        GENERATE   RVEXPO(1,96)
        ASSIGN     1,FN(JTYPE),PH
AAA     ASSIGN     2,FN(GROUP),PH
NEXT    ENTER      MH1(PH1,PH2)
BBB     ADVANCE    RVEXPO(1,MH2(PH1,PH2))
        LEAVE      MH1(PH1,PH2)
CCC     LOOP       2,NEXT
        TABULATE   PH1
        TERMINATE
        GENERATE   4800
        TABULATE   TJOBS
        TERMINATE  1
        DO   &I=1,5
        RESET
        START      5
        ENDDO
        END
```

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 22
VARIABLES and EXPRESSIONS.
THE PRINT BLOCK

Arithmetic in GPSS

We have been doing arithmetic in GPSS/H when it was needed without commenting on it. This is because arithmetic is allowed in the operands of the various blocks and is done in a logical manner. Thus, when we has a block such as ADVANCE 2*100 it was not necessary to point out that the time on the FEC for the transaction was 200. However, it is now necessary to formally go through the steps done in performing arithmetic because there are certain cautions that must be observed. These have to do with the original integer nature of calculations in GPSS. Other versions of GPSS do not allow arithmetic expressions in operands.

Arithmetic is accomplished in GPSS/H by using SNA's together with various arithmetic operations. These operations are:

+ addition
- subtraction (both unary and binary subtraction
/ division
*** multiplication**
@ modular division

We have been using all of the above except for modular division when needed. Modular division can come in quite handy. It is defined as the remainder when two number are divided. For example, 7 @ 3 is 1; 9 @ 9 is 0 (no remainder); 8 @ 12 is 8. An example of it use might be:

```
ASSIGN 2,RN1@3+1,PH
```

would assign to parameter 2 a number from 1 to 3 with equal probability.

An arithmetic expression can be placed in an operand or referenced in a manner similar to referencing functions. This might be done in GPSS/H, for example, when a particular expression is referenced multiple times and so can save considerable time in writing the code. Referencing of expressions is done by defining the expression using either a VARIABLE or a FVARIABLE expression. The form is:

```
    (name) VARIABLE (expression)
```

or `(name) VARIABLE (expression)`

Numbers can be used for variables. So,

```
TOMMY VARIABLE   3PH+FR(MACH)/200-Q(WAIT)
1     VARIABLE   X(FIRST)@X(SECOND)+QM(NEXT)
CALC  FVARIABLE  3+Q(LAST)*F(MACH)-R(TUGS)
```

are possible ways to define variables TOMMY and 1. Notice that no spaces are allowed in expressions where, in other programming languages such as Fortran, blank spaces are recommended for clarity. A blank space in this case will terminate the expression.

Referencing variables is done by:

```
V(name) (or, V$name)
```

When an expression is referenced, it is evaluated and the result is returned. Evaluation of expression follows the usual rules found in other programming languages. This means from left to right with multiplication, division and modular division having precedence over addition and subtraction. Parentheses are used for grouping and clarity. Whatever expression is innermost in nested parentheses is done first. Inner parentheses have preference over outer parentheses.

If the variable expression is defined by a VARIABLE statement, only integer calculations are done. All division is integer division, i. e., the result is truncated. However, the final result is a decimal value. This decimal value must be .0 due to the nature of having done integer calculations. Thus, the variable

```
JERRY VARIABLE   3/2+1
```

would return a value of 2.0 when referenced by V(JERRY). With FVARIABLES the expression is evaluated by doing floating point calculations. If necessary, non-decimal values are converted to floating point values. So, if the variable JERRY above had been defined as:

```
JERR FVARIABLE  3/2+1
```

The value returned would have been 2.5.

If an expression is used in an operand, then, in general, integer calculations are performed unless

floating point results are indicated. This can be done using floating point savevalues or parameters. Also, GPSS/H has two built in functions to handle arithmetic calculations where one wants to specify either fixed point or floating point calculations. These are FIX for fixed point conversion and FLT for floating point conversion. Once you specify FLT in an expression for a single SNA, the whole expression is evaluated as though it was to be done using floating point calculations. Consider the examples,

```
a) ADVANCE 3/2+1
b) ADVANCE FLT(3)/2+1
```

In a), the delay time is 2.0 but in b), the delay time is 2.5. However,

```
a) SAVEVALUE FIRST,FLT(3)/2+1
b) SAVEVALUE SECOND,FLT(3)/2+1,XL
c) SAVEVALUE THIRD,3/2+1,XL
```

In a), FIRST has a value of 2 (there would be a compiler warning to the effect that a truncation has taken place as the result of the expression is 2.0 and then conversion takes place). In b), the value of SECOND is 2.5 since the savevalue is specified as being floating point. In c), the value of THIRD is 2.0

The PRINT Block

It is possible to have statistical information sent to the report while the program is being run. This statistical information is the SNA's associated with a particular entity at the time the information is sent to the report. This is done using a PRINT block. When a transaction enters this block all the statistics associated with the specified entity (or entities) are sent to the report file. The form of the PRINT block is

```
PRINT   A,B,C,D
```

A is the lower limit of the entity range (often omitted)
B is the upper limit of the entity range (often Omitted)
C is the family name of the entity to be printed out
D used to be a printer directive but is now ignored. It will not be used here.

If A and B are omitted, all the statistics for the entity class are printed out.

Some examples of the PRINT block are:

```
a) PRINT 1,3,XL
```

```
b) PRINT ,,Q
c) PRINT 3,7,F
d) PRINT ,,MH
e) PRINT 2,2,X
```

In a), the floating point savevalues 1 to 3 are printed out.

In b), all the queue statistics are printed out.

In c), statistics for facilities 3 to 7 are printed out.

In d), all the half word matrices are printed out.

If you use labels for an entity as is the common case, it is not possible to have only selected one printed out. Since a PRINT block will add output to the normal GPSS/H report every time a transaction passes through it, caution must be taken in using it. It is mostly used for de-bugging purposes and then only when the program is run for a limited time.

Some other possible entities that can be used in the PRINT block are:

AMP	all ampervariables
B	current and total block execution counts
C	absolute and relative clock values
F	facilities
LG	all logic switched that are in a set position
MB,MH,ML,MX	various matrix values
N	current and total block execution counts
Q	queue statistics
RN	random number stream
S	storage statistics
T	table statistics
W	current and total block execution counts

Notice that if B, N or W is used, the statistics sent to the report are identical.


Return on CONTENS

Designed by Vyacheslav V. Franchuk

e-mail: franchuk@pent200.podol.khmelnitskiy.ua

John R. Sturgul

CHAPTER 23
Boolean variables

The TEST block and the GATE block have been used to allow the programmer to control the flow of transactions through the program blocks. When the transactions enters a TEST block or a GATE block, depending on the type it is, the transaction may be delayed until some condition is true, it may pass through to the next sequential block or it may be routed to another part of the program. This type of delay situations can be increase greatly by the use of Boolean variables. These allow the programmer to specify user supplied logic conditions to control the flow of transactions through a system. As such, they can be used to model very complex situations that require many different conditions to be satisfied. For example, a plane attempting a landing at a distant airport might need to meet the conditions: Is the airport open? Is the runway clear? Is there room in the hanger for the particular type of plane?, etc.

In Chapter XXX, two types of variables were introduced, fixed point and floating point. A third type of variable used in GPSS is known as a Boolean variable. This is a variable that is defined by the programmer. It will have only one of two values. These are either 0 or 1. Just as with other variables, the Boolean variable will have an associated expression, called a logical expression, which is evaluated. The value of the expression will be either 1 (true) or 0 (false).

Boolean expressions are made up of SNA's or entities connected by one or more of the following:

1. Relational Operators
2. Boolean Operators
3. Logical Operators

These will be covered next.

1. Relational (Comparison) Operators

These were introduced with the TEST and SELECT blocks. For convenience, they will be repeated here. When they are used in the TEST block or the SELECT block they are used alone, but in Boolean expressions they must have single apostrophes on either side:

Relational Operator	Meaning

'G'	**greater than**
'GE'	**greater than or equal**
'E'	**equal**
'NE'	**not equal**
'LE'	**less than or equal**
'L'	**less than**

Some Boolean expressions illustrating the above are:

```
a) Q(TOM)'E'Q(BILL)
b) X(TEST1)'L'X(TEST2)
C) FR(MACHA)'G'250
```

In a), if the queue length of the queue TOM is equal to the queue length of the queue BILL, the expression is true; otherwise, the value is 0.

In b), the fullword savevalue TEST1 must be less than the fullword savevalue TEST2 in order for the expression to be equal to 1.

In c), the utilization of the facility MACHA must be greater than 250 for the expression to be true. (Recall that the utilization of a facility is expresses in parts per thousand).

GPSS/H also allows the following symbols to be used as alternates:

Operator	Equivalent
'G'	>
'GE'	>=
'L'	<
'LE'	<=
'E'	=
'NE'	!=

Thus,

```
Q(TOM)'E'Q(BILL)
```

above could have been written:

```
Q(TOM)=Q(BILL)
```

Both forms will be used in this book.

2. Boolean Operators

The real power of Boolean variables comes from using Boolean operators to connect relational operators. There are three Boolean operators in GPSS/H. These are AND, OR and NOT. These are used by inserting these actual words between expressions that are enclosed in parentheses. The value of each is identical to similar operators in languages such as Fortran. Thus, the Boolean operator AND returns a true result only when the value of the expressions on both sides of it are true. Thus,

(true)AND(true) is true or 1
(false)AND(true) is false or 0
(true)AND(false) is false or 0
(false)AND(false) is false or 0

The operator OR returns a true result if either or both of the values of the expressions it true. Thus,

(true)OR(true) is true or 1
(false)OR(true) is true or 1
(true)OR(false) is true or 1
(false)OR(false) is false or 0

NOT inverts the value of an expression. Thus,

NOT(true) is false or 0
NOT(false) is true or 1

Some examples are as follows:

Assumptions

```
Q(TOM) = 3
```

```
Q(BILL) = 2
X(FIRST) = 5
X(SECOND) = 6
PH1 = -4
```

Boolean expressions:

```
a) (Q(TOM)'LE'4)AND(PH1'G'-5)
b) (Q(BILL)'E'2)OR(X(FIRST)'E'6)
c) (X(SECOND)'LE'X(FIRST))AND(PH1'G'0)
d) (Q(TOM)'G'2)AND(NOT(X(SECOND)'E'6)
```

a) and b) are true (value 1) but c) and d) are false (value 0).

Alternate symbols that can be used for OR and AND are "+" for OR and "*" for AND. Since these symbols are also used for arithmetic operations, this means that one cannot do addition or multiplication in Boolean expressions. Thus, if one wishes to do any addition or multiplication in a Boolean expression, one has to do this in another variable and reference it in the Boolean expression. For example,

```
ONE   VARIABLE   X(FIRST)+X(SECOND)-PH2
(Bollean expression) (Q(TOM)>=2)OR(V(ONE=0)
```

The Boolean expression will be true if the queue at TOM is equal to or greater than 2 or if the variable ONE is equal to 0.

The choice of "+" and "*" for OR and AND is also confusing because it is so easy to look at the plus sign and mentally associate it with AND. However unfortunate this is, it is a part of the GPSS language. This is a carry over from the early versions of GPSS. There is no symbol that can be used for NOT as this was not a feature of the early versions of GPSS.

The previous examples could have been written as:

```
a) (Q(TOM)'LE'4)*(PH1'G'-5)
b) (Q(BILL)'E'2)+(X(FIRST)=6)
c) (X(SECOND)'LE'X(FIRST))*(PH1'G'0)
d) (Q(TOM)'G'2)*(NOT(X(SECOND)'E'6))
```

Since it is much more logical to write out AND, OR or NOT, this will be the practice to do so in this book. Many of the parentheses used above are not needed as we shall shortly learn.

3. Logical Operators

It is possible to use logical operators to reference various entities in GPSS. A logical operator will check on the status of an entity condition. If this check is true, the value is 1, otherwise, 0. The logical operators are SNA's. Some of them are as follows:

Logical Operator	Condition Referenced
FU(name) (or F(name))	is the facility in use?
FNU(name)	is the facility not in use?
SE(name)	is the storage empty?
SNE(name)	is the storage not empty?
SF(name)	is the storage full?
LS(name)	is the logic switch set?
LR(name)	is the logic switch reset?

The above entities referenced by the logical operator could have been numbers, in which case the reference is with parentheses is optional. GPSS/H also supports the older of referencing these logical operators using a single $ sign. Thus,

```
LS$FIRST and LS(FIRST)
```

are the same.

Some examples of these operators are:

```
a) SNF(TUGS)
b) LR(STOP12)
c) F(MACH1)
d) LS6
```

In a), is the storage of TUGS is not full, the value is 1.

In b), if the logic switch STOP12 is reset, the value is 1.

In c), if the facility MACH1 is in use, the value is 1.

In d), the logic switch 6 is referenced. If it is set, the value is 1.

By combining relational operators, Boolean operators and logical operators a great deal of

complex situations can be easily modeled. For example, consider the following:

```
(LR(GOIN))AND(Q(WAIT)'LE'3)AND(FNU(MACH2)
```

In order for this to be true, the following conditions must all be true:

a) the logic switch GOIN must be reset
b) the queue at WAIT must be less than or equal to 3
c) MACH2 must be free.

Referencing Boolean Variables

Boolean variables are defined and referenced in much the same way as other variables. The general form is:

```
(name) BVARIABLE (expression)
```

It is possible to have a Boolean variable with a number for a label. One references the Boolean variable by BV(name) or BVn where (name) is the label or n the number. Thus, one might have something like

```
TEST E    BV(STOPIT),1
```

When a transaction arrives at this TEST block, the value of the Boolean variable STOPIT is determined. Unless it has the value 1, the transaction cannot move to the next sequential block. Instead it is kept on the CEC and scanned again when a re-scan is made.

It is also possible to reference Boolean variables by use of the single dollar sign, "$". Thus, the example above could have been written:

```
TEST E   BV$STOPIT,1
```

Rules for Evaluation of Boolean Expressions

The rules for evaluation of Boolean expressions are as follows:

1. Logical operators and relational operators have preference in a left to right order.
2. The operators AND, OR and NOT are evaluated in a left to right order.

Parentheses can (and should) be used for clarity. When used, whatever is in the innermost parenthesis is evaluated first.

Example 23.1 (from Schriber)

A port is used to load tankers with crude oil. The port can load up to 3 tankers simultaneously. Tankers arrive every 11 ± 7 hours and are of three different types. The relative frequency of the various types and their loading time requirements, are given in Table 23.1.

Table 23.1 Tanker specifications

type	rel. freq.	loading time, hours
1	.25	18 ± 2
2	.55	24 ± 3
3	.20	36 ± 4

There is one tug at the port. Tankers of all types require the services of this tug to move into a berth and later to move out of a berth. Furthermore, the area experiences frequent storms, and no berthing or deberthing of a tanker can take place when a storm is in progress. When storms occur, they last 4 ± 2 hours. The time between the end of one storm and the onset of the next follows the exponential distribution and have a mean value of 48 hours. When a tug is available and no storm is in progress, berthing or deberthing activity takes about 1 hour.

A shipper is considering bidding on a contract to transport oil from the port to the United Kingdom. He has determined that 5 tankers of a particular type would have to be committed to this task to meet contract specifications. These tankers would require 21 +/- 3 hours to load oil at the port. After loading and deberthing, they would travel to the UK, off-load the oil, return to the port for reloading, etc. Their round-trip travel time, including off-loading, is estimated to be 240 ± 24 hours.

Build a GPSS model to measure in-port residence times of the proposed additional tankers, as well as the three types of tankers which already use the port. Use a 3 - year simulation to estimate the average time in the port for all types of ships.

Solution

The program to do the simulation is as follows:

```
      SIMULATE
      STORAGE    S(BERTH),3
TYPE  FUNCTION   RN1,D3
.25,1/.8,2/1,3
SPRED FUNCTION   PH1,L4
1,2/2,3/3,4/4,3
MEAN  FUNCTION   PH1,L4
1,18/2,24/3,36/4,21
GOIN  BVARIABLE  (SNF(BERTH))AND(FNU(TUG))AND(LR(STORM))
GOOUT BVARIABLE  (FNU(TUG))AND(LR(STORM))
1     TABLE      M1,20,10,9
2     TABLE      M1,20,10,9
3     TABLE      M1,40,10,9
4     TABLE      MP3PL,20,10,9
SHIP  GENERATE   ,,,5,,4PH,4PL          FIVE BIG OIL TANKERS
      ASSIGN     1,4,PH                 CALL THESE #4
      ADVANCE    48*N(SHIP)-48          SPACE OUT ARRIVALS
      MARK       3PL                    MARK ARRIVAL TIME
      TRANSFER   ,PORT                  SEND TO PORT
TIMES GENERATE   11,7                   OTHER SHIPS ARRIVE
      ASSIGN     1,FN(TYPE),PH          DETERMINE WHICH TYPE
PORT  TEST E     BV(GOIN),1             CAN SHIPS ENTER PORT?
      SEIZE      TUG                    USE TUG BOAT
      ENTER      BERTH                  USE A BERTH
      ADVANCE    1                      TUG BERTHS A SHIP
      RELEASE    TUG                    FREE THE TUB
      ASSIGN     2,FN(SPRED),PH         DETERMINE SPREAD
      ADVANCE    FN(MEAN),PH2           LOAD A SHIP
      TEST E     BV(GOOUT),1            CAN SHIP LEAVE?
      SEIZE      TUG                    USE TUG BOAT
      ADVANCE    1                      DE-BERTH THE SHIP
      RELEASE    TUG                    FREE THE TUG
      LEAVE      BERTH                  FREE THE BERTH
      TABULATE   PH1                    MAKE TABLE OF RESIDENCE TIMES
      TEST NE    PH1,4,CYCLE            FILTER OUT SHIPS
      TERMINATE  OTHER                  SHIPS LEAVE
CYCLE ADVANCE    240,24                 TO UK AND BACK
      MARK       3PL                    MARK TIME BACK
```

```
      TRANSFER   ,PORT                  ARRIVE AT PORT
      GENERATE   ,,,1                   DUMMY TRANSACTION
NEXT  ADVANCE    RVEXPO(1,48)           STORM ON ITS WAY
      LOGIC S    STORM                 STORM ARRIVES
      ADVANCE    4,2                   EVERYTHING DOWN
      LOGIC R    STORM                 STORM OVER
      TRANSFER   ,NEXT                 WAIT FOR NEXT STORM
      GENERATE   8760*3                3 YEARS PASS
      TERMINATE  1                     END OF SIMULATION
      START      1,NP
      PUTPIC     LINES=11,FILE=SYSPRINT,(N(CYCLE),N(TIMES),_
                 N(NEXT),FR(TUG)/1000,SR(BERTH)/1000,_
                 TB1,TB2,TB3,TB4)
0 |=============================================|
  | NUMBER OF LARGE OIL TANKERS        = ****     |
  | NUMBER OF OTHER SHIPS IN PORT      = ****     |
  | NUMBER OF STORMS IN THE YEAR       = ****     |
  | UTILIZATION OF THE TUG BOAT        =    *.*** |
  | UTILIZATION OF THE BERTHS          =    *.*** |
  | AVERAGE TIME IN PORT FOR SHIP #1   =  ***.**  |
  | AVERAGE TIME IN PORT FOR SHIP #2   =  ***.**  |
  | AVERAGE TIME IN PORT FOR SHIP #3   =  ***.**  |
  | AVG. TIME IN PORT FOR OIL TANKERS  =  ***.**  |
  |=============================================|
      END
```

The output from the program is:

```
|=============================================|
| NUMBER OF LARGE OIL TANKERS        =  454     |
| NUMBER OF OTHER SHIPS IN PORT      = 2393     |
| NUMBER OF STORMS IN THE YEAR       =  531     |
| UTILIZATION OF THE TUG BOAT        =    0.216 |
| UTILIZATION OF THE BERTHS          =    0.968 |
| AVERAGE TIME IN PORT FOR SHIP #1   =   47.34  |
| AVERAGE TIME IN PORT FOR SHIP #2   =   53.77  |
| AVERAGE TIME IN PORT FOR SHIP #3   =   65.02  |
| AVG. TIME IN PORT FOR OIL TANKERS  =   49.52  |
|=============================================|
```

The tug boat is certainly not being overused. the berths are being used 96.8% of the time so the port is nearing capacity. The oil tankers spend about 50 hours in the port. Loading takes an average of 23 of these hours so they are spending more than an additional 1 full day. (around 27 hours). Whether or not this is acceptable will depend on the economics of the situation.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 24
The BUFFER block

To fully appreciate the way a GPSS program works, it is important to always understand the way transactions are moved on the various chains. Recall that once a transaction is moved by the processor, it will be moved forward as far as it can until one of either three things happen to it:

a) it is terminated
b) it is put on the FEC
c) it is blocked.

When one of the above happens, the processor will start a re-scan. So far, this is always what we have wanted to have happen. However, there are times when it is necessary to start a re-scan of the CEC before the active transaction has come to a rest. This can perhaps best be understood by considering a short example.

Suppose a delivery company has 5 trucks in its fleet. Each has a daily availability of 70% because some are used in other parts of the company or some are in for maintenance and/or repairs. At the start of each day, it is desired to determine how many are available. If 5 are available, they will be allocated a certain way; if 4 are available, they will be allocated a different way, etc. The program lines for this might be (it will be explained why the PRIORITY 0 block is added):

```
HOWMNY FUNCTION  RN1,D2
.7,OKTRK/1,BYEBYE
WHERE FUNCTION   NTRUCK,L5
1,BLOCKA/2,BLOCKB/3,BLOCKC/4,BLOCKD/5,BLOCKE
      GENERATE   ,,,5,1
      PRIORITY   0
      TRANSFER   ,FN(HOWMNY)
OKTRK SAVEVALUE  NTRUCK+,1
      ( blank for now )
      TRANSFER   ,WHERE
```

Five truck transactions are generated at time 0. Each has priority 1. Once a transaction leaves the GENERATE block, its priority is reduced to 0. It then enters the block TRANSFER ,FN(HOWMNY). The function HOWMNY will determine if the truck transaction will be available for the day or not. If it is to be available, the next block, OKTRK SAVEVALUE NTRUCK+,1

keeps a count of the number of trucks that are available. At the pointt when a transaction enters the TRANSFER ,WHERE block you want the transaction to be transferred to different parts of the program depending on how many are available. These are indicated by the block labels, BLOCKA, BLOCKB, etc. Suppose the first transaction leaves the GENERATE block and is routed to the block labeled OKTRK. It will increment the count of NTRUCK to 1 and then go to the next sequential block. If this is the TRANSFER ,WHERE block, it will be routed to BLOCKA which is where you want the transactions to go if the is only one truck available. Since there might be more trucks available, this is incorrect. The way to handle this is to have a re-scan of the CEC at this point and move the other truck transactions which are residing in the GENERATE block. This is exactly what the BUFFER block does. Its general form is:

BUFFER

There are no operands. When a transaction enters this block, it causes a re-scan of the CEC.

Let us see how this works for the example above. Imagine that there is a BUFFER block where it says (blank for now). The processor starts a re-scan. The transaction in the BUFFER block is a part of this scan. Its priority is 0. The second truck transaction in the GENERATE ,,,5,1 block has a priority of 1 so it is moved first. Thus, the first transaction will be held at the BUFFER block until this second transaction is moved forward. How far then is it moved? Just as far as the BUFFER block when a re-scan is begun. Now, the third transaction is moved forward from the GENERATE ,,,5,1 block as far as the BUFFER block. This is repeated for until all 5 of the transactions have left the block. Now, when each transaction enters the TRANSFER ,WHERE block, the correct count of the available trucks in the system for the day is used to allocate them.

Example (from Schriber)

A library does not have an open stack policy. Instead people fill out slips and take them to the circulation desk where a clerk goes into the stacks to find the book. People who want to check out books arrive at the desk in a Poisson stream at a mean rate of 30 per hour. Each person wants to check out only 1 book, which is always available. The number of clerks working is a variable of the problem. Each clerk will pick up as many as 4 slips. The times for these operations are:

1. The time required to pick up a slip is small.
2. A one-way trip to the stacks takes 1 ± .5 minutes.
3. The time to find 1, 2, 3, or 4 books is normally distributed with means of 3, 6, 9, and 12 minutes respectively. The std. dev. is 20% of the mean.
4. When a clerk returns from the stacks, the rest of the checkout procedure is completed on a first come, first served basis. Thus, the clerk gives the book first to the customer who gave her the slip first.
5. If 2 or more clerks are idle when a customer arrives at the checkout desk, the clerk who

was been idle the longest is the one who serves the customer.
6. If two or more people are waiting for service, when 2 or more clerks become available, the clerks do not divide the work. Instead, one clerk picks up as many slips as possible.

The program needs to have logic to do the following:

1. Each clerk can pick up a maximum of 4 slips.
2. Each clerk needs to know exactly how many slips she has.
3. Customers must be able to identify with the customers.
4. Clerks must be able to identify with the customers.
5. Customers can leave the desk one at a time.

The program to do the simulation is:

```
      SIMULATE
      INTEGER    &I
SLIP  EQU        10,L
      STORAGE    S(BUSY),3
DELAY TABLE      M1,6,1,26
SLIPS TABLE      X(COUNT),1,1,5
DOUBL BVARIABLE  (X(COUNT)'E'4)OR(W(WAIT)'E'0)
BLOCKA GENERATE  ,,,&I                NUMBER OF CLERKS WORKING
      ASSIGN     1,N(BLOCKA),PH       NUMBER EACH CLERK
BLOCKB TEST G    W(WAIT),0            ANY WAITING?
      ENTER      BUSY                 YEP, BECOME BUSY
      SAVEVALUE  COUNT,0              NUMBER OF SLIPS PICKED UP
      SAVEVALUE  CLERK,PH1            CLERK = NUMBER OF CLERK
      LOGIC S    SLIP                 OPEN GATE FOR SLIPS
      BUFFER                          RE-SCAN
      ASSIGN     2,X(COUNT),PH        PAR. 2 = NUMBER OF SLIPS
      TABULATE   SLIPS
      ADVANCE    1,.5                 WALK TO STACKS
      ADVANCE    RVNORM(1,PH1,PH1/5)*3    GET THE BOOK(S)
      ADVANCE    1,.5                 RETURN FROM STACKS
BLOCKC ADVANCE   2,1                  FINISH THE CHECKOUT
      LOGIC S    PH1                  OPEN GATE
      BUFFER
      LOOP       2,BLOCKC             LOOP AROUND
      LEAVE      BUSY
      TRANSFER   ,BLOCKB              GO WAIT AGAIN
```

```
      GENERATE   RVEXPO(1,2),,,,10    CUSTOMERS ARRIVE
WAIT  ADVANCE                        THEY WAIT
      GATE LS    SLIP
      ASSIGN     1,X(CLERK),PH
      SAVEVALUE  COUNT+,1
      TEST E     BV(DOUBL),1,BYPAS
      LOGIC R    SLIP
BYPAS GATE LS    PH1
      LOGIC R    PH1
      TABULATE   DELAY
      TERMINATE  1
      DO   &I=3,6
      CLEAR
BLOCKA GENERATE  ,,,&I
      STORAGE    S(BUSY),&I
      START      500,NP
      PUTPIC     LINES=6,FILE=SYSPRINT,(&I,SR(BUSY)/1000,_
                 TB(DELAY),TB(SLIPS))
0 |=============================================|
  | NUMBER OF CLERKS             = ***          |
  | UTILIZATION OF CLERKS        =   *.***      |
  | AVERAGE DELAY OF CUSTOMERS   =  **.**       |
  | AVERAGE NUMBER OF SLIPS      =   *.**       |
  |=============================================|
      ENDDO
      END
```

The output from the program is as follows:

```
|=============================================|
| NUMBER OF CLERKS             =   3          |
| UTILIZATION OF CLERKS        =   0.946      |
| AVERAGE DELAY OF CUSTOMERS   =  13.80       |
| AVERAGE NUMBER OF SLIPS      =   2.10       |
|=============================================|

|=============================================|
| NUMBER OF CLERKS             =   4          |
| UTILIZATION OF CLERKS        =   0.926      |
```

```
| AVERAGE DELAY OF CUSTOMERS  =  13.65          |
| AVERAGE NUMBER OF SLIPS     =   1.71          |
|===============================================|

|===============================================|
| NUMBER OF CLERKS            =   5             |
| UTILIZATION OF CLERKS       =   0.890         |
| AVERAGE DELAY OF CUSTOMERS  =  13.74          |
| AVERAGE NUMBER OF SLIPS     =   1.43          |
|===============================================|

|===============================================|
| NUMBER OF CLERKS            =   6             |
| UTILIZATION OF CLERKS       =   0.870         |
| AVERAGE DELAY OF CUSTOMERS  =  14.31          |
| AVERAGE NUMBER OF SLIPS     =   1.33          |
|===============================================|
```

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 25
The SPLIT block

Transactions have been placed in our models by the GENERATE block. In fact, this is the only way to create original transactions. However, once a transaction is in a model it is possible to make clones of the original transactions. These clones will normally be identical to the original transactions, although they can be made to differ. As far as being identical to the original transactions, the clones will always be identical in the priority level and the time of entry (their Mark Time). This latter point is worthy of noting. If the original transaction entered the model at time 2050 and at time 3500 a new transaction was cloned, the clone has a Mark Time of 2050, not 3500. The clones will normally have the same number and type of parameters as the original but it is here that the clones can be made to differ. How to do this will be discussed below.

The block that creates these clones is the SPLIT block. The form to create identical transactions is:

```
SPLIT n,(label)
```

where n is the number of clones to create
label is the block label the transactions are routed to.

When a transaction enters a SPLIT block, the n identical transactions are created and leave the block one at a time (incrementing the block count as they leave). These are all routed to the block whose label is specified in the B operand of the SPLIT block. The original transactions is not routed to this block but goes to the next sequential block. In fact, this original transaction is moved before the clones are. Some examples of the SPLIT block are;

```
a) SPLIT 1,DOWN1
b) SPLIT 10,UPTOP
```

In a), one new transaction is created and sent to the block with the label DOWN1. In b), 10 new transactions are created and sent to the block with the label UPTOP. In both cases, the original transactions are routed to the next sequential block.

Often it is desired to have the original transaction and the clones routed to the same block. This can be done by making the next sequential block the one where the clones are sent to, i. e.,

```
SPLIT 3,NEXT1
```

```
NEXT1 (next block)
```

Here the original and the 3 clones are sent to the same block.

These split blocks can come in very handy in programming problems where a single unit comes along and several things have to be done one different parts of the unit and these are to be done simultaneously. For example, suppose a partially completed car comes along an assembly line. At this point one person will make an adjustment to the front, another to the rear and suppose a third bolts on a part. It will be convenient to split the car transaction by making two clones and have the clones worked on separately. Only when all three transactions (the original and the two clones) are finished can the car be moved along. How to do this will be covered later.

Example 25.1

In a manufacturing process parts come along an assembly line every 4 ± 1.2 minutes. An overhead crane is used to lift them from the line to another section where they will be worked on further. It takes 2 ± .8 minutes to load and transport the parts. The crane then must return to the original position. This takes 1.6 ± .3 minutes. Give the GPSS code to model this segment.

Solution

```
GENERATE    4,1.2          PARTS COME ALONG
QUEUE       WAIT           JOIN QUEUE
SEIZE       CRANE          USE THE CRANE
DEPART      WAIT           LEAVE THE QUEUE
ADVANCE     2,.8           MOVE PART TO NEW SECTION
SPLIT       1,DOWN1        CREATE CLONE
ADVANCE     1.6,.3         MOVE CRANE BACK
RELEASE     CRANE          FREE CRANE
TERMINATE                  REMOVE TRANSACTION
DOWN1       (next section)
```

Notice that after the crane moved the part, the part formed a clone which is sent to the block DOWN1. The original part transaction entered the block ADVANCE 1.6,.3. This represents the travel time for the crane to return to the original position. When it leaves this block, it release the CRANE. Only then can another part be moved by the crane.

Example (from Schriber)

A certain machine uses a type of part which is subject to periodic failure. Whenever the in-use part fails, the machine must be turned off. The failed part is then removed, a good spare part is installed if available, or as soon as one becomes available, and the machine is turned on again. Failed parts can be repaired and used again (forever).

The lifetime of a part is normally distributed with a mean of 350 hours and a standard deviation of 70 hours. It takes 4 hours to remove a failed part from the machine. The time required to install a replacement part is 6 hours. Repair time for a failed part is normally distributed with mean and standard distribution of 8 and 0.5 hours respectively.

The machine operator himself is responsible for removing a failed part from the machine, and installing a replacement part in its place. There is a repairman who is responsible for repairing failed parts. The repairman's duties also include repair of items routed to him from another source. These other items arrive in a Poisson stream with a mean interarrival time of 9 hours. Their service time requirements is 8 +/- 4 hours. These other items have a higher priority than the failed parts used in the machine of interest.

Each hour the machine is down costs the company $160. Each spare part cost the company $125/week (40 hours) to keep in stock. Determine how many spares to have. Suppose your answer is X spare parts. Use a spread sheet to determine when the costs to the company is such that it can consider switch X - 1 spares and then to X + 1 spares.

Build a GPSS model of the system to see how the efficiency of the machine increase with the number of spare parts. Run for 40 years, assuming 40 hour weeks. (GENERATE 40*52*40)

Solution

The program to do the simulation is as follows:

```
      SIMULATE
      INTEGER    &I
      GENERATE   ,,,1
WAIT  SEIZE      MACH1
      ADVANCE    RVNORM(1,350,70)
      RELEASE    MACH1
      ADVANCE    4
      SPLIT      1,FETCH
      SEIZE      FIXER
      ADVANCE    RVNORM(1,8,.5)
      RELEASE    FIXER
```

```
      SAVEVALUE  PART+,1
      TERMINATE
FETCH TEST G     X(PART),0
      SAVEVALUE  PART-,1
      ADVANCE    60
      TRANSFER   ,WAIT
      GENERATE   RVEXPO(1,9),,,,1       OTHER PARTS COME
      ADVANCE
      SEIZE      FIXER
      ADVANCE    8,4
      RELEASE    FIXER
      TERMINATE
      GENERATE   52*40*40
      SAVEVALUE  COST,(1-FLT(FR(MACH1))/1000)*40*160+(&I-1)*_
                 FLT(125),XL
      TERMINATE  1
      DO   &I=1,5
      CLEAR
      INITIAL    X(PART),&I-1
      START      1,NP
      PUTPIC     LINES=8,FILE=SYSPRINT,(&I-1,FR(MACH1)/1000,_
                 FR(FIXER)/1000,FC(MACH1),FC(FIXER),_
                 XL(COST))
0 &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =  ***          &
& UTILIZATION OF MACHINE      =    *.***      &
& UTILIZATION OF FIXER        =    *.***      &
& NUMBER OF BREAKDOWNS        =  ***          &
& NUMBER OF TIMES FIXER USED  = ****          &
& EXPECTED COST               = ****.**       &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
      ENDDO
      END
```

The output from the program is:

```
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =    0          &
& UTILIZATION OF MACHINE      =    0.539      &
```

```
& UTILIZATION OF FIXER        =    0.901    &
& NUMBER OF BREAKDOWNS        =  126        &
& NUMBER OF TIMES FIXER USED  = 9334        &
& EXPECTED COST               = 2952.99     &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =    1        &
& UTILIZATION OF MACHINE      =    0.737    &
& UTILIZATION OF FIXER        =    0.898    &
& NUMBER OF BREAKDOWNS        =  176        &
& NUMBER OF TIMES FIXER USED  = 9306        &
& EXPECTED COST               = 1811.27     &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =    2        &
& UTILIZATION OF MACHINE      =    0.762    &
& UTILIZATION OF FIXER        =    0.903    &
& NUMBER OF BREAKDOWNS        =  181        &
& NUMBER OF TIMES FIXER USED  = 9406        &
& EXPECTED COST               = 1773.52     &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =    3        &
& UTILIZATION OF MACHINE      =    0.843    &
& UTILIZATION OF FIXER        =    0.894    &
& NUMBER OF BREAKDOWNS        =  201        &
& NUMBER OF TIMES FIXER USED  = 9282        &
& EXPECTED COST               = 1381.89     &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
& NUMBER OF SPARE PARTS       =    4        &
& UTILIZATION OF MACHINE      =    0.815    &
& UTILIZATION OF FIXER        =    0.885    &
& NUMBER OF BREAKDOWNS        =  191        &
& NUMBER OF TIMES FIXER USED  = 9185        &
```

```
& EXPECTED COST                = 1681.27     &
&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
```

With no spares, the utilization of the machine is only 53.9%. With a single spare, the utilization increases to 73.7%. With 4 spares the utilization is up to 81.5%. The minimum cost associated with the spares is $1,381.89 for 3 spares.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua

CHAPTER 26
The ASSEMBLY SETS and the ASSEMBLE block

All transactions belong to different groups known as assembly sets. When a transaction is created via the GENERATE block, it is assigned its own assemble set. Once a transaction is assigned its own assembly set, it remains there until it leaves the system. These sets are not numbered or named so a person cannot refer to them. Only after a transaction leaves the system and re-enters later can it be assigned a different assembly set. When only a single transaction is in an assembly set, there is not much of interest with the set. It is when there are more than one transaction belonging to a particular assembly set that is of interest to the programmer.

When a transaction enters a SPLIT block, the cloned transactions belong to the same assembly set as the original. Even if these cloned transactions themselves enter SPLIT block, the newly cloned transactions belong to the same assembly sets. Several blocks are used with the concept of assembly sets. The first is the ASSEMBLE block.

The ASSEMBLE Block

The ASSEMBLE block acts in a manner the opposite to the SPLIT block. The SPLIT block clones new transactions into the system and the ASSEMBLE block removes them. The form of it is:

```
ASSEMBLE    n
```

n is the number of transactions to be removed from the system. When a transaction enters the ASSEMBLE block it is delay there until other members of its assembly set also arrive in the block where each is removed. Only when the counter, as given by the operand n is reached, is the original transaction allowed to move to the next sequential block. It is immaterial if the first transaction to arrive at the ASSEMBLE block is the original uncloned (parent) transaction or not. The first transaction in the block is the one that is allowed to move on.

For example, when a transaction enters the block

```
ASSEMBLE    2
```

It will be delayed until two other transactions from its assembly set also enter the block and are subsequently destroyed. A program might have a set of blocks such as:

```
        SPLIT    2,DOWNA
        ......   ......
        ......   ......
DOWNA   ......   ......
        ......   ......
        ASSEMBLE 2
```

The SPLIT b The SPLIT block creates two clones when the original transaction enters it. Later, two of the transactions in the same assembly set are removed from the system. The transaction that goes to the sequential block after the ASSEMBLE block is not necessarily the original transaction. It is possible to have the same ASSEMBLE block working on more than one assembly set. For a given assembly set it is possible to have assembling operations being done at more than one ASSEMBLY block. Thus,

```
        SPLIT     6,DOWN1
        ......    .......
DOWN1   ......    .......
        ......    .......
        TRANSFER  ,FN(AWAY)
BLOCKA  ASSEMBLE  2
        .......   .......
        .......   .......
BLOCKB  ASSEMBLE  3
        .......   .......
        .......   .......
```

The block TRANSFER ,FN(AWAY) might send some of the cloned transactions to the block labeled BLOCKA while others might go to the block labeled BLOCKB.

Example 26.1

A worker needs to fill a case with boxes of toys. Each case can hold 10 boxes. The worker needs to do the following:

a) take an empty box from a stack and being it to a bench.
b) pick up a toy, inspect it and place it a small box.
c) place an inspection tag in the box with the toy.
d) when there are 10 boxes in the case, close the case and place it on an assembly line.

These operations take the following times:

a) 15 ± 8 seconds
b) mean 110 sec., std. of 20 seconds (normally dist.)
c) mean of 30, exponentially dist.
d) 25 ± 10 seconds

Simulate for 5 shifts work if each shift is 400 minutes long. Determine how many cases are loaded onto the assembly line by the worker.

The program to do the simulation is:

```
      SIMULATE
      INTEGER    &I
      GENERATE   ,,,1
BACKUP SEIZE     PERSON
      SPLIT      1,BACKUP
      ADVANCE    15,8
      ADVANCE    RVNORM(1,110,20)
      ADVANCE    RVEXPO(1,30)
      ADVANCE    25,10
      RELEASE    PERSON
      ASSEMBLE   10
      SAVEVALUE  CASES+,1
      TERMINATE
      GENERATE   60*60*6
      TERMINATE  1
      DO   &I=1,5
      CLEAR
      START      1,NP
      PUTPIC     LINES=6,FILE=SYSPRINT,&I,X(CASES)
0 |=============================================|
  |                                             |
  |           <<< DAY NUMBER ** >>>             |
  |                                             |
  |   NUMBER OF CASES LOADED IN THE DAY = ****  |
  |                                             |
  |=============================================|
      ENDDO
      END
```

The output is as follows:

```
|=============================================|
|                                             |
|           <<< DAY NUMBER  1 >>>             |
|                                             |
|    NUMBER OF CASES LOADED IN THE DAY =  12  |
|                                             |
|=============================================|

|=============================================|
|                                             |
|           <<< DAY NUMBER  2 >>>             |
|                                             |
|    NUMBER OF CASES LOADED IN THE DAY =  11  |
|                                             |
|=============================================|

|=============================================|
|                                             |
|           <<< DAY NUMBER  3 >>>             |
|                                             |
|    NUMBER OF CASES LOADED IN THE DAY =  11  |
|                                             |
|=============================================|

|=============================================|
|                                             |
|           <<< DAY NUMBER  4 >>>             |
|                                             |
|    NUMBER OF CASES LOADED IN THE DAY =  11  |
|                                             |
|=============================================|

|=============================================|
|                                             |
|           <<< DAY NUMBER  5 >>>             |
|                                             |
```

```
|    NUMBER OF CASES LOADED IN THE DAY =  12  |
|                                             |
|=============================================|
```

As can be seen, the number of cases loaded each shift is either 11 or 12.

Example (from Schriber)

PERT Diagrams

The network shown in Figuree 26.1 represents a series of subprojects which must be carried out to complete an overall project. A pair of circles (nodes) connected by a directed line segment is used to depict each particular subproject. For example, node 1 is connected to node 2, depicting what is called subproject 1-to-2. Each directed line segment is labeled to show how many people and how many time units are required to perform the corresponding subproject. Subproject 1-to-2 requires 4 people, then, and takes 14 ± 6 time units to complete.

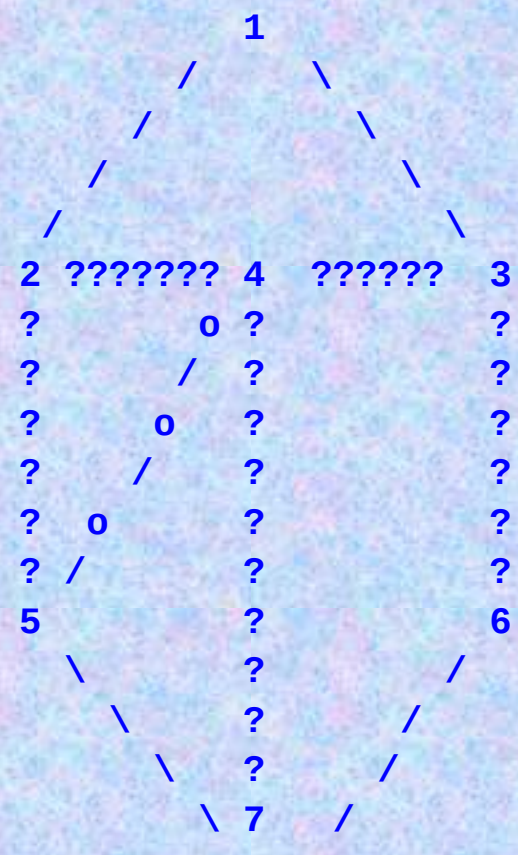


Figure 26.1 PERT diagram.

The times and people needed for each project are given in Table 26.1

Table 26.1 Times and people needed for each job.

project	people needed	time
1 -to- 2	4	14 ± 6
1 -to- 3	3	20 ± 9
2 -to- 4	3	10 ± 3
2 -to- 5	5	18 ± 4
3 -to- 4	2	22 ± 5
3 -to- 6	1	25 ± 7
4 -to- 7	4	15 ± 5
4 -to- 5	0	no time
5 -to- 7	2	8 ± 3
6 -to- 7	4	10 ± 3

Figure 26.1 also displays the precedence constraints, indicating which subprojects must be completed before other subprojects can be started. For example, subprojects 2 -to- 4 and 3 -to- 4 must be completed before subproject 4 -to- 7 can be started. Similarly, subproject 5 -to- 7 cannot be initiated until subproject 2 -to- 5 is finished, and until the subprojects leading into node 4 have been completed.

In projects of this kind, it is of interest to know how much time is required to complete the overall project. Build a GPSS model to simulate the undertaking of the overall project shown in Figure 26.1.

Solution

The program to do the simulation is given below:

```
      SIMULATE
      INTEGER    &I
      STORAGE    S(WORKER),5
RTIME TABLE      M1,25,25,20
      GENERATE                    PROVIDE A TRANSACTION
      GATE LR    NEXT1            CAN IT ENTER THE SYSTEM?
      LOGIC S    NEXT1            YES, SHUT GATE
```

```
NODE1  SPLIT      1,SUB13          WORK ON 1 - 3 AND 1 - 2
SUB12  ENTER      WORKER,4         4 WORKERS FOR 1 - 2
       ADVANCE    14,6             DO WORK
       LEAVE      WORKER,4         WORKERS FREE
NODE2  SPLIT      1,SUB24          WORK ON 2 - 4
SUB25  ENTER      WORKER,5         5 WORKERS FOR 2 - 4
       ADVANCE    18,4             DO THE WORK
       LEAVE      WORKER,5         FREE THE WORKERS
NODE5  ASSEMBLE   2                ARE JOBS DONE?
SUB57  ENTER      WORKER,2         WORK ON 5 - 7
       ADVANCE    8,3              DO THE WORK
       LEAVE      WORKER,2         FREE THE WORKERS
       TRANSFER   ,NODE7           CHECK ON OTHER JOBS
SUB24  ENTER      WORKER,3         WORK ON 2 - 4
       ADVANCE    10,3             DO THE WORK
       LEAVE      WORKER,3         FREE THE WORKERS
NODE4  ASSEMBLE   2                CHECK ON OTHER JOBS
       SPLIT      1,NODE5          BEGIN WORK ON 4 - 7 AND 4 - 5
SUB47  ENTER      WORKER,4         PROVIDE WORKERS
       ADVANCE    15,5             DO THE WORK
       LEAVE      WORKER,4         FREE THE WORKERS
NODE7  ASSEMBLE   3                CHECK ON OTHER JOBS
       TABULATE   RTIME            TABULATE TIME TO DO JOB
       LOGIC R    NEXT1            START NEW JOB
       TERMINATE  1                JOB DONE
SUB13  ENTER      WORKER,3         WORK ON 1 - 3
       ADVANCE    20,9             DO THE WORK
       LEAVE      WORKER,3         FREE WORKERS
NODE3  SPLIT      1,SUB34          BEGIN WORK ON 3 - 6 AND 3 - 4
SUB36  ENTER      WORKER           WORK ON 3 - 6
       ADVANCE    25,7             DO THE WORK
       LEAVE      WORKER           FREE THE WORKER
NODE6  ENTER      WORKER,4         WORK ON 3 - 6
       ADVANCE    10,3             DO THE WORK
       LEAVE      WORKER,4         FREE THE WORKERS
       TRANSFER   ,NODE7           SEE IF OTHER JOBS DONE
SUB34  ENTER      WORKER,2         WORK ON 3 - 4
       ADVANCE    22,5             DO THE JOB
       LEAVE      WORKER,2         FREE THE WORKERS
```

```
      TRANSFER   ,NODE4           CHECK ON OTHER JOBS
      DO   &I=5,12               START DO LOOP
      CLEAR                      CLEAR FOR NEXT RUN
      RMULT      777             RANDOM NUMBER GENERATOR
      STORAGE    S(WORKER),&I    PROVIDE WORKERS
      START      500,NP          DO FOR 250 JOBS
      PUTPIC     LINES=6,FILE=SYSPRINT,(&I,TB(RTIME),_
                 SR(WORKER)/10,SM(WORKER))
0 |=========================================|
  | NUMBER OF WORKERS PROVIDED  =  ***      |
  | AVERAGE TIME TO DO JOB      =  ***.**   |
  | UTILIZATION OF WORKERS      =   **.**%  |
  | MAXIMUM NO. WORKERS NEEDED  =   **      |
  |=========================================|
      ENDDO
      END
```

The results of the simulation are:

```
|=========================================|
| NUMBER OF WORKERS PROVIDED  =    5      |
| AVERAGE TIME TO DO JOB      =  117.33   |
| UTILIZATION OF WORKERS      =   72.01%  |
| MAXIMUM NO. WORKERS NEEDED  =    5      |
|=========================================|

|=========================================|
| NUMBER OF WORKERS PROVIDED  =    6      |
| AVERAGE TIME TO DO JOB      =   98.54   |
| UTILIZATION OF WORKERS      =   71.42%  |
| MAXIMUM NO. WORKERS NEEDED  =    6      |
|=========================================|

|=========================================|
| NUMBER OF WORKERS PROVIDED  =    7      |
| AVERAGE TIME TO DO JOB      =   81.97   |
| UTILIZATION OF WORKERS      =   73.60%  |
| MAXIMUM NO. WORKERS NEEDED  =    7      |
|=========================================|
```

```
|============================================|
| NUMBER OF WORKERS PROVIDED  =    8         |
| AVERAGE TIME TO DO JOB      =   62.92      |
| UTILIZATION OF WORKERS      =   83.97%     |
| MAXIMUM NO. WORKERS NEEDED  =    8         |
|============================================|

|============================================|
| NUMBER OF WORKERS PROVIDED  =    9         |
| AVERAGE TIME TO DO JOB      =   66.39      |
| UTILIZATION OF WORKERS      =   70.74%     |
| MAXIMUM NO. WORKERS NEEDED  =    9         |
|============================================|

|============================================|
| NUMBER OF WORKERS PROVIDED  =   10         |
| AVERAGE TIME TO DO JOB      =   65.14      |
| UTILIZATION OF WORKERS      =   64.89%     |
| MAXIMUM NO. WORKERS NEEDED  =   10         |
|============================================|

|============================================|
| NUMBER OF WORKERS PROVIDED  =   11         |
| AVERAGE TIME TO DO JOB      =   58.76      |
| UTILIZATION OF WORKERS      =   65.35%     |
| MAXIMUM NO. WORKERS NEEDED  =   11         |
|============================================|

|============================================|
| NUMBER OF WORKERS PROVIDED  =   12         |
| AVERAGE TIME TO DO JOB      =   58.76      |
| UTILIZATION OF WORKERS      =   59.90%     |
| MAXIMUM NO. WORKERS NEEDED  =   11         |
|============================================|
```

With only 5 workers available, the time for the project took 117.33 time units. The addition of one more worker reduced this to 98.54 time units. When there were 11 workers, the time is reduced to 58.76 time units. In fact, when another worker is added, there is no change in the average time to

complete the job. In fact, if an infinite number of workers are applied, the average time to do the job is reduced only to 58.56 time units.

Return on CONTENS

Designed by Vyacheslav V. Franchuk
e-mail: franchuk@pent200.podol.khmelnitskiy.ua