**Homework “1” (aka h1) - CSC148, Prof Mitchell, Sept 14, 2020**

**‘Extending the first queueing model’**

**Due date: On/Before midnight Mon. Sept 21**

**Added Section#5 Submission Requirements & Simplified Section#4, Sept 17**

Course module P1 covered the simplest form of 1-server system having a service wait queue. *Note:* gpss refers to each single server as a “facility”.

*h1 implements significant extensions of SAMPLE1.GPS.*

*The new blocks/statements will be used in all remaining gpss work in CSC148.*

**#1 More on using gpss**

**Invoking gpss:** the sequence: File 🡪 New 🡪 Model (NOT Text File) 🡪 OK

invokes an editing session in which you can input a new “model”, that is, a new gpss-formatted source file (*caution: not a pure .txt file*).

Input all source lines for a complete gpss model. Assume you name your source file h1.gps, save it using the usual Windows file menu item Save As.

gpss files can be stored in any folder, independent of gpss .exe location.

gpss File main menu item has the simple sub-menus Open, Close, Save and Save As.

While still in an editing session for h1.gps, and assuming you are ready to execute the source file, select gpss menu items

Command 🡪 Create Simulation. This creates a “journal” file (by default, named h1.sim) that traces major events (start, end, suspend, etc.) during simulation execution. If error messages do not appear, you should see 2 lines indicating successful model translation, like:

07/17/17 16:53:34 Model Translation Begun.

07/17/17 16:53:34 Ready.

gpss now waits for a command (to execute the model, or some other command)

Choose: Command 🡪 START, and a small popup window contains START 1.

The “1” is a default initialization value for the (termination count (tc)) control variable. Recall that the tc value specifies the number of customers the model must process.

Input the tc value for a given run. Click OK and the run begins.

Save/delete, etc. any of the files produced by the run, as needed. For example, each time a simulation run is executed, another x.sim journal file is created, and another report (.rpt) file is also generated.

Note that gpss-generated x.gps source files are not raw unformatted .txt files; they should only be opened from within a gpss session; however, report files and x.sim files are Windows textfile formatted files.

**#2 Extending the single-barber simulation**

In the /homework subdirectory of course homePage, copy: **h0BaseModel\_148\_f20.gps**

In gpss use File - -> Open to modify it, as specified below, for this assignment.

Remember to first adjust the preamble documentation to match this assignment.

{ ***You must use the source code statement formats shown in* h0BaseModel\_148\_f20.gps**

***for all models in this course, adjusting as necessary, per model*** }

*Note! h1 specifications for the t.u. are different from h0BaseModel\_148\_s18.gps.*

*DES is general/flexible since the t.u. can be any magnitude in any given model – choose the t.u. value most convenient (second, year, hour, …) for your application.*

**gpss statement case sensitivity**

You can intermix upper and lower case in gpss language tokens for more readability (Example: identifier oneLABEL is the **SAME** as ONELABEL).

The following illustrates this:

Source model:

generate 1

oneLABEL advance 1

ONELABEL advance 1

terminate 1

Translates with errors:

09/03/17 21:04:54 Model Translation Begun.

09/03/17 21:04:54 Line 3, Col 1. Multiple label definitions:

09/03/17 21:04:54 ONELABEL advance 1

09/03/17 21:04:54 \*\*\*\* Model Translation Aborted \*\*\*\*

**The following specifies how to extend your copy of h0BaseModel\_148\_f20.gps.**

This doc assumes the server is named barber (you can name it anything).

**Add** a **CLEAR** statement as the first block or statement following the Preamble documentation. This statement removes any residual or state values (if any) persisting in gpss memory from previous gpss executions in this session.

*This is a gpss coding convention for rest of course.*

Use EQU statements (place them after the CLEAR and before any other

statements or blocks) to define a symbol for each constant numeric or

string scalar value. EQU is the way to define the value of a symbol in gpss.

Example EQU statement along with a comment:

interArrMean EQU 10 ; ia mean is 10 minutes (assuming t.u. = 1 minute)

(You know the benefits of symbolic code from previous programming experience;

the same applies for gpss).

You must code an EQU statement for each of the Operands of the GENERATE and ADVANCE blocks. Thus, as illustrated above, specify GENERATE A and B Operand values **10 and 6,** and ADVANCE A and B Operand values **7 and 3** as symbols.

We want the interarrival and service durations to be comparable to SAMPLE1.GPS.

Therefore, the specified A and B operand values above, integers < 10, mean that the t.u. in this homework is: 1 t.u. is 1 minute

**Add** one **EQU** statement and one **RMULT** statement following this EQU to seed your model’s random number generation as follow:

Label your EQU statement and for the X Operand, choose any six-digit unsigned integer seed value. Then code an **RMULT** statement with A Operand = the label of the EQU statement. (RMULT will NOT need a label).

The EQU and RMULT statements above are better coding compared with

RMULT dddddd *(dddddd is your chosen 6-digit seed)*

The EQU and RMULT, as coded above, cause an internal gpss random number generator (named “RN1”) to draw random numbers for calculating interarrival and service durations and other internal runtime values needed by the simulator.

The seed is involved in the initialization for RN1’s random number stream.

8 built-in random number generators are available in any gpss model.

*Notes on DES model seeding - Executing a given simulation model two times using the same generator seeds will produce IDENTICAL RESULTS. This is a great benefit because a given model’s results will be completely reproducible.*

*Such results are peer/expert reviewable, and publishable if they are identical – this reproducibility criterion is the basis for the “scientific method”.*

*By contrast, two executions of a model that differ only in random seeding will produce different results, either slightly different, or more noticeably different, depending on the model.*

In general, a QUEUE/DEPART pair can be placed at any positions in source code – it depends on which stats need to be measured at what region in the model.

The original source code, SAMPLE1.GPS, covered in module P1 has A operand Barber for the QUEUE, SEIZE, DEPART and RELEASE block with the same name.

Although this is legal gpss code, it is poor programming practice for different block types to share operand names – it is confusing.

**Therefore, replace the A operand of the QUEUE and DEPART with w\_barber**

Each QUEUE/DEPART pair gathers stats for each entity during the time that entity was in that region of the source code ; these stats are automatically reported in .rpt file QUEUE section (*no need to write code for this reporting*).

*Note: When first using gpss, QUEUE/DEPART block pairs are confusing because of their gpss block names. A QUEUE block does not cause the queueing of an entity to some processing queue. In M1Q the SEIZE block determines customer queueing.*

**Add** another **QUEUE** with Operand res\_time, before or after QUEUE w\_barber, and corresponding **DEPART** res\_time after the RELEASE block. This pair of blocks gathers various stats for the entire residence duration for each customer.

The gpss TABLE statement is used to specify a histogram representation of some distribution of values during model execution.

We will use QUEUE w\_barber to gather service wait population data (i.e customer counts), and use QUEUE res\_time to gather residence duration data (i.e. customer residence duration in the model).

**Add** two **TABLE** statements anywhere in the definitions above the GENERATE block.

Generally, TABLE Operands specify the structure for a histogram of the distribution of some value of interest.

Study the TABLE documentation, and see that the statement is coded:

label TABLE A,B,C,D

The TABLE label is required, and can be any meaningful identifier.

The 4 operands A,B,C,D are all required – they specify the structure/shape if the histogram of a particular distribution.

First, for QUEUE w\_barber, the table’s A operand must be q$w\_barber. q$ is a gpss function call that returns the number of entities (here customers) that currently are waiting for service.

Operand B specifies the largest value in the first histogram class (that is, the right-most value of the first cell. Operand C specifies the size of each equal-sized class. You choose Operand D to get the required final histogram display. D Operand (“frequency class” is same as cell, corresponding to each bar in a histogram display) must be a positive integer (not a symbol).

Second, for QUEUE res\_time the table’s A Operand should be the function M1.

“M1” is a gpss built-in/pre-defined function (with no arguments) that returns an entity’s residence time in S at any place where M1 is referenced.

What is left is specifying the model place at which QUEUE/DEPQRT stats are captured. This is the purpose of the gpss TABULATE block.

**Add,** after the QUEUE w\_barber block, the block **TABULATE** **tableName1,** where tableName1 is the label of your corresponding TABLE statement**.** This block increments by 1 the histogram cell value for the current queue size (population).

Also **Add**, after DEPART res\_time, **TABULATE** **tableName2,** where tableName2 is the label of your corresponding TABLE statement for QUEUE res\_time

Each histogram produced by gpss is rendered in two ways:

1. Automatic:: Each gpss .rpt file contains a tabular display of the contents of all histogram cells (Caution: thus, a histogram with many cells (500, 2000 … ?) could create a huge .rpt file.
2. Optional:: Display each TABLE histogram graph by the menus path: Window -> Simulation Window -> Table Window -> select desired histogram.

Finally, for h1, the remaining blocks of h0BaseModel\_148\_f20.gps: RELEASE,

second DEPART, and TERMINATE 1 blocks and Operands are unchanged.

**Aside – a note on facility state**

*Notice the AVAIL. display in a .rpt file, and its value (always =1 in models covered so far). At any model time, if a facility is “up”, it is available (AVAIL value 1), otherwise, it is “down”/non-operational/unavailable (AVAIL. value 0).*

**#3 GENERATE and ADVANCE block Operand*s***

In h1, the GENERATE operands A specifies the mean of the interarrivals of customers, and the B specifies the “half interval” (gpss terminology), i.e,

(1/2)\*(max interarrival value – min interarrival value). This notation is specific to uniform distributions. This gives successive interarrival times: a1, a2, a3, … and each ai is uniformly random in [10-6,10+6]. Thus, interarrivals times are no smaller than 4 t.u., and no larger than 16, with average value 10.

Similarly, service durations specified by A and B Operands of ADVANCE statement are random and uniformly distributed in [7-3,7+3].

*Note: the value of a uniformly-distributed random variable on [a,b] can be either an integer or a real value with a fraction significant to 6 decimal places in the continuous domain [a, b]. (gpss numeric precision limits possible distribution values)*

Other distributions, used in many advanced apps, such as Poisson, exponential, lognormal, etc., have non-symmetric probability density/mass functions.

**#4 Required results from each run**

**You must do 2 model runs, as specified in section #5 each run having an RMULT with a different 6-digit seed value. (Simply change EQU’s X Operand for each run). Different seeds produce (in this model, slightly) different run results.**

**Start each run via command START 10000 (ten thousand). *The source code is the SAME* *for each run, except that each run uses a different seed value.***

[*This model scenario, with “no rest or days off”, does not model any real barber shop. The reason for the large service count is to confirm that some of the model results stats do seem to converge to theoretical values such as r*]

**#5 Submission requirements**

**In ONE file, place the following items on order:**

1. **A copy of your source code**
2. **First 4 lines of the .rpt file for Run#1 ONLY**
3. **The FACILITY section (2 lines, including the FACILITY label line) and QUEUE section (3 lines, including the QUEUE label line) of the .rpt file for Run#1**
4. **The FACILITY section (2 lines, including the FACILITY label line) and QUEUE section (3 lines, including the QUEUE label line) of the .rpt file for Run#2**
5. **The histogram graph corresponding to the QUEUE/DEPART res\_time for Run#1 ONLY (Recall how to display it: near the end of Section #2), and use any snipping tool to get the histogram image.**

**email ONE attachment that includes items 1.-5. above to billmit@ecs.csus.edu**

**#6 How TABLE and TABULATE work together**

All DES simulator systems have a way to code a description of histogram or frequency distribution. gpss calls such a structure a ‘TABLE’, and other languages call them something else (no programming standardization ever happened across simulation languages). However, gpss has one of the SIMPLEST ways to specify generation of histogram cell entries.

In the tr labeled “demoHisto” below, a histogram is specifieded using two entities, a TABLE statement (technically, a gpss “statement”, not a “block”) and a TABULATE block. The code skeleton looks like:

myHisto TABLE Operands A,B,C, and D

*{ <=* *TABLE statements can be placed anywhere in a gpss model, but it is good code organization to place them with the EQU and other definitions at top of code }*

:

demoHisto GENERATE user-chosen Operands

:

TABULATE myHisto

:

TERMINATE Operand

In general, there can be >1 TABULATE within one tr.

After tr “t” is generated, when t enters the TABULATE block, TABLE Operand A (it can be any legal gpss expression) is evaluated to value v, and the histogram cell where value v lies is incremented (by default) by 1.

Example: suppose that c had a total model residence time of 20 t.u., but that when c entered the (assume only) TABULATE block, c had been in the model for 14.2 t.u.

Then the cell frequency count is incremented by 1 in myHisto according to the cell to which 14.2 belongs.

This is an error because it means that the TABULATE block is in the wrong place in the source code.

Thus, where a TABULATE is placed in the code matters.

Note - TABLE A Operand values might not be time values – they are app-dependent. Operand A can be any expression. Example: in network packet traffic, A values could be packet activity counts at network nodes.