7 Programming considerations.

Commencing with release 08.00 there is a built-inn facility in the 360/370 SIMULA System for recording program behaviour during the execution. A user can obtain useful insight into the program dynamics by requesting results of various measurements to be output.

In the following the user will be given full orientation about the tools available for improving performance of the SIMULA programs. It is not necessarily intended for a thorough study - on the contrary, it is hoped that a user interested in using this facility will find all sufficient information in paragraphs 7.1 and 7.4, eventually supported by a glimpse of the example. However, should she/he have any enquiries it should be possible to find full answers in the rest of the text.

7.1 Objectives and design principles.

The purpose of this facility (referred to as the turner in the rest of this text) is to provide a sufficient insight into the dynamics of execution in order to reduce its cpu-time and storage requirements. To this end special focus is put on supervising the two most critical issues:

- data transmissions realized through various kinds of assignments (the most frequently encountered operation)
- allocation of data structures (a complex operation dictated by the dynamic nature of SIMULA).

The character of this facility, however, suggests that it can also be useful in the program debugging process.

Since the collection of the various statistics during the program execution represents whatever small but nevertheless a certain overhead the facility has been designed in such a way that its function (and thus the overhead) can be completely eliminated at the user's will. Therefore the user should carefully consult paragraph 7.4 below to make sure of correct activation of the various tuner components.

7.2 Scope of the facility.

Even though useful in itself the bulk of this programming aid is just an offspring of other program development tools SYMBDUMP and TRACE (see separate documentation). In particular the latter and specifically its data flow monitoring subsystem are of primary importance for understanding the dynamics of the program behaviour. However, since these tools are basically designed for program debugging, their use for program tuning can be somewhat cumbersome.

In contrast to these facilities, the tuner concentrates on giving just the resulting contour of the dynamic profile of program execution. This image is created by several components outlined below.

7.2.1 Assignment counter

This is a global counter which is increased by one for each assignment statement carried out by the system.

Included here are:

- explicit value- and reference-assignments affecting user declared data items;
- implicit assignments in for-statements and those of parameter transmission to procedures and classes by value or by reference,
- text value assignments implicit to text PUT-attributes and the corresponding OUT-procedures of files (assignments to image subtexts).

Excluded (i.e. not recorded in the assignment counter) are the following assignments:

- those leading to alteration of system declared data items e.g. text position indicator setting, manipulation of SUC and PRED through SIMSET procedures, manipulation of the SQS through SIMULATION utilities and parameter transmission for standard procedure calls,
- parameter transmission of value and reference parameters to virtual, formal and external procedures,
- all assignments in SIMULA program modules which were compiled with option avoiding assignments recording.

The current value of the global assigment counter is available during the program execution through use of an external procedure TRACECNT (cf. TRACE documentation). This can be conveniently used for dynamic initiating/terminating of other program execution supervising facilities. The final value is output at the end of execution in the Run Time System trailing line. This value tends to be a fair measure of the volume of the performed computation.

7.2.2 Frequency monitor.

This is a control flow monitor updating local counters associated with both explicit and implicit assignment statements. Each time an assignment statement (of the kind affecting assignment counter, see above) is executed its local counter is increased by one so that at the termination of the program the values recorded in the local counters reflect fairly precisely the dynamic profile of the execution.

The contents of the local counters can be obtained at the end of execution both in numeric and graphic form. Furthermore indication is given of the most frequently executed as well as the void assignments. These tables give a deep insight into the program dynamics and specifically turn attention to the apparent execution time bottlenecks. Be aware though that a flat histogram does not necessarily imply the most efficient algorithm.

7.2.3 Data structure register.

This is a device for recording the number of generated instances of each data structure defined in the program. These include:

- block instances of various kinds (however, only those which are user defined appear in the final account).
- user and system generated text objects.
- all declared arrays and arrays passed as parameters by value.

Not counted are:

- system generated control blocks (drivers and event notices since these are at least partly reused).
- system generated storages for holding temporary results which may be quite many but are unfortunately not under direct user control.
- static instances of standard procedures, buffers and other accessories necessary for the operating system interface because these do not take space in the working storage.

While the current value of a particular object counter serves at each time as a unique identification of the next generated instance of this kind, the final list of these counters is most valuable information for the assessment of program storage requirements. In addition the ratio of the respective data structures in the total program storage demand is estimated. Obviously, this kind of information is a more direct hint on how to improve program performance than just a number of store collapses.

With the considerable overhead of SIMULA storage management in mind, one can achieve drastic reductions of cpu-time usage by minimizing e.g. the number of generated procedure instances. However, one more aspect which must be considered here is the prevailing way in which a procedure is invoked (direct or remote call, virtual, formal procedure etc.).

7.3. Output format of the respective components.

All the information provided by this facility appears on the standard printfile SYSOUT appended to the user program produced printout. With the exception of the histogram part of the frequently monitor the output can be intercepted also on a teletype compatible device with line length of 80 characters.

With the exception of the global assignment count the output is organised by modules: the first come execution frequency counters and the list of allocated data structures concerning the main program alone and then follow corresponding tables for the separately compiled modules used in the program (if any) in turn.

When organizing the output modulewise the following rules are observed:

- it is the maximum assignment frequency within a particular module that is used for forming out the histogram and for marking the lines of specific interest in the module.

- the data structure counts indicate the outcome of the complete program execution. Consequently also the logical partitioning of all allocated storage on the fragments taken by the respective data structures shown in the last (%) column is related to the complete execution.

- Obviously text and array objects may be allocated in the course of execution of any module. However their statistics are only given globally and appear together with the dynamic profile of the main program.

As regards the particular format of the respective tuner components see illustration in the paragraph 5 and also consider the remarks below.

7.3.1 Final assignment count.

Final assignment count appears (if requested, see par. 4) at the RTS trailing line in the form:

<count> ASSIGNMENTS RECORDED

Be aware that the true value can be greater than the figure given in case that one or more separately compiled modules used in the program had assignment counting suppressed.

7.3.2 Assignment frequencies.

Assignment frequencies (if registered) appeared under the title:

DYNAMIC PROFILE OF THE PROGRAM EXECUTION BASED
ON DATA FLOW MONITORING

The output consists basically of two parts:

- a numeric table of absolute frequencies given for the respective assignments in the given module,
- a histogram (adjacent to the table) showing schematically relation between the maximum frequency on a line and the maximum frequency within the module.

Note that the original source text is not reproduced as is sometimes customary in these reports. Instead references are made to its respective lines through the line numbers.

Several further details may be of interest:

- o respective frequencies for up to 7 (first) assignments are registered on each line - the user must alone match the figures given with the corresponding source listing (identified in the report),
- o the maximum frequency on a line appears in a clearly marked column next to the line number,
- o the maximum frequency over the whole module is marked by a sequence of exclamation marks at the left of the line number.

Finally note that the following convention is used when generating these tables: in default only lines where there has been at least one assignment executed appear in the table. Optionally however, the user can require a report on every single line. If this is the case the following rules apply:

- o just the line number alone is given for lines without any executable code,
- o the line number followed by a colon with the rest of the line blank signifies a line with executable code not containing any assignment,
- o all lines containing assignment statements of which none was performed in the course of the program execution are marked by a sequence of question marks to the left of the line number.

7.3.3 Generated data structures.

List of the generated data structures appears under the title:

USER GENERATED DATA STRUCTURES

Listed in the table are various block patterns defined in the module in question, some instances of which were generated during the program execution. For each block pattern, the following information is given:

- pattern identification (its kind and commencing line number in the source (text),
- total number of generated instances,
- size (in bytes) of instance. Note that this is the true size i.e. includes space for user data, the necessary system overhead, block counter location plus eventual padding to meet the storage allignment requirements (if any),
- share of the total program storage demand attributed to this particular block kind (in percent rounded to the nearest integer).

Following the table the total sum of all generated block instances is given together with an indication of how large a portion of the total storage used this represented. (Note that when estimating total storage use only true store collapses are considered i.e. the effect of garbage collections occurring as a side effect e.g. various program development utilities are disregarded.)

If the related module is the main program the total number of generated text and array objects together with the indication of how large a portion of the total storage used these represented is also given in percent.

Note that the percent figures will not sum up to 100%: the discrepancy is accounted for by the system generated data structures (drivers, event notices and temporary and standard objects) which do not appear anywhere in this table.

Finally it is worth mentioning that also this table is compressed to the block patterns of which no instances were generated at all are suppressed in default. Since their appearance may have som value for program documentation, the user can optionally request them to be output together with nonempty lines.

7.4 Control.

The production of the tuner printouts is under full user control by means of the compiler and the runtime system options.

The following combinations are required for output of the respective items:

total number of assignments registered during program execution will appear if:

- the program was compiled with SYMBDUMP>3
- and run with the TEST option.

assignment frequency table (and the histogram) will appear if:

- the program (module) was compiled with SYMBDUMP>3,
- the program was run with SYMBDUMP>3
- and the program terminated with return code zero (i.e. without any RT-error).

the generated data structures are listed if:

- the program was compiled with SYMBDUMP>1
- and run with SYMBDUMP>3.

Note that default settings at SIMULA system distribution are SYMBDUMP=0 for compiler and SYMBDUMP=1 for the runtime system so that they have to be either permanently reset at the installation or overriden at individual runs in order to activate the tuner.

As regards the expansion of the tables to their full length (i.e. including also the void entries which are suppressed in default, see par. 3) the following conventions apply:

The specification of the RTS option SYMBDUMP must be followed immediately by

- o the seguence AA to force "All Assignments" to appear, or by
- o the sequence AB to force "All Blocks" to appear, eventually by
- o the letter A alone get All assignments and blocks,

e.g.

```
//EXEC SIMCLG,PARM.SIM='SYMBDUMP=4',PARM.GO='SYMBDUMP=4AB'
```

will cause the full table of user defined block patterns to appear while the assignment frequency table which will also be generated will include only non-empty entries.

Note that the best way how to alter default values of the compiler and the RTS option SYMBDUMP is through the use of the macros SIMCDF and SIMRDF which are part of the system delivery. This would however not suffice for the expansion of the tuner tables where the letters A, AA or AB can only be specified at execution.

7.5 Example

Let us suppose that the following program is compiled with SYMBDUMP=4
(shown below is an extract of the compilation listing):

```
begin class BOARD(E,F,RIMCHAR); integer E,F; character RIMCHAR;
      begin character array C(@:E,@:F); integer I,J,M,N;

            procedure PRINT;
            for I:=@ step 1 until E do
                begin for J:=@ step 1 until F
                      do Outchar(C(I,J));
                      Outimage
                end***PRINT***;

            procedure READ;
            for I:=1 step 1 until M do
                begin Inimage;
                      for J:=1 step 1 until N do
                          if Inchar=' '
                             then C(I,J):=' ' else C(I,J):='*';
                end***READ***;

            M:=E-1; N:=F-1;

            for I:=@ step 1 until N do
                C(@,I):=C(E,I+1):=RIMCHAR;
            for J:=@ step 1 until M do
                C(J+1,@):=C(J,F):=RIMCHAR;
      end***BOARD***;

      BOARD class LIFE;
      begin integer procedure NEIGHBOURSOF(I,J); integer I,J;
            begin integer T;
                  if C(I-1,J-1)='*' then T:=T+1;
                  if C(I-1,J  )='*' then T:=T+1;
                  if C(I-1,J+1)='*' then T:=T+1;
                  if C(I  ,J-1)='*' then T:=T+1;
                  if C(I  ,J+1)='*' then T:=T+1;
                  if C(I+1,J-1)='*' then T:=T+1;
                  if C(I+1,J  )='*' then T:=T+1;
                  if C(I+1,J+1)='*' then T:=T+1; NEIGHBOURSOF:=T;
            end***NEIGHBOURSOF***;

            procedure WRITE(GENERATION); integer GENERATION;
            begin Outtext("GENERATION NO "); Outint(GENERATION,6);
                  Outimage; PRINT
            end***WRITE***;
      end***LIFE***;
```

Continued on the next page....

....Continued from the previous page:

```
      procedure SWAP;
      begin TEMP:-OLDB; OLDB:-NEWB; NEWB:-TEMP end***SWAP***;

      ref(LIFE) OLDB,NEWB,TEMP;      character C1;
      integer Linesperpage,P,Q,GENS,Z, LIFELENGTH;
      external real procedure BALAS;

      LIFELENGTH:=BALAS(Inint,Inint);

      Outtext("INITIAL STATE"); Outimage;
      P:=Inint;    Q:=Inint;   Linesperpage:=64;

      OLDB:-new LIFE(P,Q,'+'); OLDB.READ;  OLDB.PRINT;
      NEWB:-new LIFE(P,Q,'+');

      for GENS :=1 step 1 until LIFELENGTH do
          begin inspect OLDB
                    do begin for P.=1 step 1 until M do
                         for Q:=1 step 1 until N do
                             begin Z:=NEIGHBOURSOF(P,Q);  C1:=C(P,Q);
                                   if Z=3
                                      then C1:='*'   else
                                   if Z=2 and C1='*'
                                      then C1:='*'   else C1:=' ';
                                   NEWB.C(P,Q):=C1;
                             end
                       end;
                NEWB.WRITE(GENS);  SWAP
          end
end
```

Note in particular the declaration of an external procedure BALAS at line 53 and its use at line 57. Let us assume that the procedure BALAS was compiled with SYMBDUMP=3 and that the program above is run with the run time option TEST and SYMBDUMP=4.

Disregarding the normal output which is of no particular interest to this demonstration the following two lines will at last appear on SYSOUT:

```
END OF SIMULA PROGRAM EXECUTION AT 17:@6:46.86
EXECUTION TIME @.@9 SEC.
RETURN CODE IS #@@@@@@@@
@@@@@ STORECOLLAPSES, DATA STORAGE USED : 9184 BYTES
21@8 ASSIGNMENTS RECORDED
```

These are the RTS trailing lines (usually prefixed by the ZYQ994 message identifier). The very last item is the final value of the global assignment counter implying in this case that altogether 2108 data transmission were recorded when executing this program. (Be aware though that this does not include assignments needed for execution of the procedure BALAS because that was compiled with SYMBDUMP less than 4).

On the next page comes the assignment frequency table and the histogram shown in Appendix A. Note that only "active" lines appear in this overview. Obviously line 8 contains the most frequently executed construction.

(As regards the for-statements with step-until elements remember that these contain two implicit assignment statements, see Common Base, section 6.2.3., case 2. These may appear on two successive lines if the for-statement is split).

The dynamic profile of the main program then continues on the next page by

USER GENERATED DATA STRUCTURES:

BLOCK PATTERN	LINE	TOTAL	SIZE	%
SUB-BLOCK	@@@2	1	64	1
PROCEDURE PRINT	@@@5	8	24	2
PROCEDURE READ	@@12	1	24	@
CLASS LIFE	@@28	2	48	1
PROCEDURE NEIGHBOURSOF	@@29	112	32	39
PROCEDURE WRITE	@@41	7	16	1
PROCEDURE SWAP	@@47	7	16	1
TOTAL NUMBER OF BLOCK INSTANCES:		138	-	46
TOTAL NUMBER OF TEXT OBJECTS :		2	-	3
TOTAL NUMBER OF ARRAY OBJECTS :		8	-	7

THE LINE NUMBERS CORRESPOND TO THE SOURCE COMPILED ON 25 NOV 198@ AT 17:@5:28.13

Finally, the very last page holds the dynamis profile (or what is available of it remember the compilation of the external mode was carried out with SYMBDUMP=3) concerning the module BALAS:

DYNAMIC PROFILE OF THE PROGRAM EXECUTION ** MODULE: BALAS **

USER GENERATED DATA STRUCTURES:

BLOCK PATTERN	LINE	TOTAL	SIZE	%
PROCEDURE BALAS	@@14	1	8@	1
PROCEDURE MAXX	@@22	3	32	1
PROCEDURE MAXT	@@29	4	48	2
PROCEDURE CHANGE	@@5@	4	32	1
PROCEDURE SOLVER	@@61	2	32	1
TOTAL NUMBER OF BLOCK INSTANCES		14	-	6

THE LINE NUMBERS CORRESPOND TO THE SOURCE COMPILED ON 25 NOV 1980 AT 17:@5:@8.13

Note here that of the total storage required for the execution of the above program approximately 52% (46+6) was taken up by various instances of which most is clearly attributed to procedure NEIGHBOURSOF altogether 112 times.

DYNAMIC PROFILE OF THE PROGRAM EXECUTION BASED ON DATA FLOW MONITORING

NOTES	LINE	MAXIMUM	FREQUENCES OF THE RESPECTIVE ASSIGNMENTS				----- H I S T O G R A M -----
	@@@6:	48	8	48			********
	@@@7:	48	48				********
!!!!!	@@@8:	288	288	288			**********
	@@13:	4	1	4			*
	@@15:	16	4	16			***
	@@17:	12	4	12			**
	@@2@:	2	2	2			
	@@22:	1@	2	1@			**
	@@23:	1@	1@	1@			**
	@@24:	1@	2	1@			**
	@@25:	1@	1@	1@			**
	@@31:	3@	3@				*****
	@@32:	41	41				*******
	@@33:	32	32				*****
	@@34:	41	41				*******
	@@35:	43	43				*******
	@@36:	32	32				*****
	@@37:	43	43				*******
	@@38:	112	35	112			**********
	@@42:	7	7	7			*
	@@48:	7	7	7	7		*
	@@57:	1	1				
	@@59:	1	1				
	@@6@:	1	1	1	1		
	@@62:	1	1	1	1	1	
	@@63:	1	1	1	1	1	
	@@65:	7	1	7			*
	@@67:	28	7	28			*****
	@@68:	112	28	112			**********
	@@69:	112	112	112	112	112	**********
	@@71:	28	28				*****
	@@73:	63	21	63			**********
	@@74:	112	112				**********
	@@77:	7	7				*

8 Bibliography.

(1) SIMULA Users Guide. Publication No. S-24.
Norwegian Computing Center, Oslo, Norway.

(2) IBM System/360 OS: Storage Estimates.

(3) IBM System/360 OS: Concepts and Facilities.
Form C28-6535.

(4) IBM System/360: Principles of Operation.
Form A22-6821-7.

(5) IBM System/360 OS: Assembler Language.
Form C28-6514-5.

(6) IBM System/360 OS: Job Control Language.
Form C28-6539-8.

(7) IBM System 360 OS: Supervisor and Data Management
Services. Form C28-6646-2.

(8) IBM System/360 OS: Supervisor and Data Management Macro
Instructions. Form C28-6647-3.

(9) IBM System/360 OS: Utilities. Form C28-6586-9.

(10) IBM System/360 OS: Linkage Editor.
Form C28-6538.

(11) IBM System/360 OS: Messages and Codes.
Form C28-6631-6.

(12) Ole-Johan Dahl, Bjørn Myrhaug, Kristen Nygaard:
SIMULA 67 Common Base Language. Publication No. S-22.
Norwegian Computing Center, Oslo, Norway.

(13) P. Naur (Ed.): Revised Report on the Algorithmic
Language Algol 60.

(14) IBM System/360 OS: FORTRAN IV library subprograms.
Form C28-6569.

(15) IBM System/360 OS: FORTRAN IV (G & H)
Programmers Guide. Form C28-6817.

Note: Page number given in references to IBM documentation are only approximates, since they may vary with the release number.