PhUSE 2016

Paper CC09

Fundamental Advice to Save Time in SAS® Programming

Mickael Borne, 4Clinics, Paris, France

ABSTRACT

This paper presents a collection of SAS tips, techniques, and advice that SAS users can implement to write and run programs faster and more efficiently. These tips and techniques come from my 15 years of experience as a SAS user and from learning from others through their presentations and papers.

Good programming practices (GPPs), regular training, and mentoring relationships can help programmers save coding time. On the other hand, offering programmers tools tailored to assist them in their current job is also a good way to reduce programming time. I provide an introduction to the SAS Function CoMPiler (FCMP procedure), which allows programmers to develop their own functions.

Managing large datasets and the resources linked to data access (I/O) are challenges for all programmers, but particularly for those working at home. On this topic, the sorting and joining of data are discussed and an introduction to the use of hash objects is proposed.

INTRODUCTION

The theme of our 2016 PhUSE conference is “How to achieve a fast track to approval?” So, what can we do in SAS programming to reduce the time to approval? Upon reflection, there are two levels on which we can act, the first is the speed at which a programmer delivers a program and the second is the speed at which a program runs.

To speed up the coding, we will discuss GPPs, programming automation with SAS Macros, of course, but I will also provide a presentation of the FCMP procedure and offer some programming advice.

To speed up the processing, we will focus on the handling of large datasets, and on the management of the reading and writing of SAS data files (SAS I/O processing), which are especially important topics for home-based workers. I will also offer some advice to help programs run faster.

SPEED UP THE CODING

All experienced programmers will tell you that, with age and experience you learn to be more efficient and faster in your coding. But, unfortunately, for me, I often lose up to one hour in programming before I realized that I did not completely understand the request, and I went in the wrong direction. In this part, we will study how to capitalize on this experience to improve programming speed and why plan is the most important thing.

THE IMPORTANCE OF GOOD PROGRAMMING PRACTICES AND TRAINING

Today, our company must produce Standard Operating Procedures (SOPs) to define GPP. It’s not ‘just’ to meet regulatory or client requirements but also to increase the robustness and reusability of the programs because GPPs are known to reduce logical and syntax errors. So, by applying GPPs, you can save maintenance time and you can facilitate the transfer of code among projects or programmers.

A SAS programmer’s dexterity is developed through training and experience. Training can improve skills and expertise, and even long-time SAS users can benefit from additional training. Because SAS programming demands ever-increasing specialization, training is the good way to stay abreast of evolving technology.

Another way to improve the dexterity of SAS programmers is to facilitate a mentoring relationship. Mentoring could be defined as an employee training system under which a senior or more experienced individual (the mentor) acts as an advisor, or coach to a less experienced programmer (the mentee). The mentor is responsible for providing support, feedback, and guidance to the individual in his or her charge. Such a relationship is really a win-win situation, with the mentee directly benefiting from mentor’s experience and the mentor taking time to answer questions he might not normally consider or take the time to answer.

PROGRAMMING AUTOMATION

Each of us knows the interests of process automation in SAS, and the time and effort taken to develop and maintain the automation, and to train others in its use is paid back after a few projects. Use of validated tools not only saves development time, but also ensures compliance with SOPs and the quality of the delivered results.

Like many things, in SAS there is more than one way to automate a process, you can of course produce a SAS program or a SAS macro program, but beyond these basics you can also use the FCMP procedure.

Under SAS there are many built-in functions that can be used to execute specific tasks. Starting with SAS 9.2, you can now write your own user-defined functions or CALL routines. Most SAS programmers know the ROUND function, for example, which rounds the first argument to the nearest multiple of the second argument. Now, with the FCMP procedure you can write your own function and use it as you do with the ROUND function (e.g. in a data step, a proc SQL). Similarly, for the well-known CALL SYMPUT routine, which assigns the value of a macro variable produced in a DATA step, you can use the FCMP procedure to write your own CALL routine.

In Appendix I, I provide an example of using of the FCMP procedure, which allows you to use variables of a SAS dataset as the first and the second arguments of a ROUND function.

PROGRAMMING ADVICE

"*Take time to plan*": Many times, we are tasked to develop automated, repeatable processes. Before starting to write the SAS code, take some time to plan your program and do not hesitate to include this as part of the comments. It will never be wasted time!

"*Execute your program step-by-step*": Even if it seems faster to run your entire SAS program all at one time, I recommend executing them step-by-step, and taking the time to open the intermediate datasets and validate the final results. Remember that just because a program runs without syntax error does not necessarily guarantee the logical correctness of the result.

"*You are not alone*": When you are up against a wall of a problem, you are likely to find a solution on the web. At the SAS community http://www.sascommunity.org/wiki/Main_Page or Lex Jansen websites http://www.lexjansen.com/ you can find many solutions from other SAS programmers.

"*Think simplification*": When you have a complex program, it is useful to extract most complex part of your code from the full program and run it locally in a temporary program. The best example is the production of a graphic. When you have created the SAS dataset containing the data needed to produce the graphic, you can save this dataset and then work it independently. It will run faster if you can execute the graphical SAS procedure alone (with minimal ODS features), rather than to executing it in the context of the full program.

"*Think modularity*": What is true for a complex IT project can also be true for your program. Grouping parts that accomplish similar things can help save maintenance time and improve the reusability of your code.

SPEED UP THE PROCESSING

I will present several choices that SAS programmers can make regarding processing efficiency. As is often the case, each option has advantages and disadvantages. Programmers need to consider all aspects of the application to make the appropriate choice, but nevertheless a program must be clear and correct before it can be fast.

HANDLING LARGE DATASETS

In pharma, we often must manage large datasets, for example in cases where data are pooled from several studies (e.g. ISS/ISE) or when we have to connect to a specific database (e.g. medical dictionary). Manipulating such large datasets can be very time consuming.

Programmers should know that, under SAS, there are several ways to join two datasets. However, performance will depend on the type of the relationship between the datasets (e.g. one-to-one, one-to-many, or many-to-many) and on their size.

One well-known technique, the MERGE statement, could be the worst in terms of performance (especially in case of many-to-many [1]). MERGE with INDEXES allows you to skip the sorting step, which could help save time in some cases. Another way to join several tables is to use more than one SET statement by either using the KEY= data set option after defining an INDEX or the POINT= option of the set statement. The technique I prefer, to the chagrin of some of my colleagues, is the SQL procedure, which allows us to perform all the types of joining we need. However, I would like to take this opportunity to present a technique that is certainly less well known but the fastest one I know of, and that is the use of the HASH object.

The HASH object is an object (not a statement) that is used in a DATA step between DATA and RUN statements. We can represent this object as a record/variable table loaded in memory. The HASH object has to be created, defined, and filled before you can access it. After the RUN statement, it will be deleted [2]. In Appendix II, I show how a HASH object is used to merge a LB dataset with a laboratory parameter dictionary.

SAS I/O PROCESSING

SAS I/O processing, the reading and writing of SAS data, can be very time consuming, especially when the SAS job has to manage large datasets (>2 Gb). One solution when you are dealing with a large amount of data, in some cases, is to avoid using the system file cache. Depending on your operating system (Direct I/O on Unix [DIO] or Scatter Read/Gather Write I/O on Windows [SGIO]) you may have to manage some I/O SAS options in order to optimize the performance [3]. However this requires access to SETINIT of SAS which is not the case for all programmers. Personally, the greater part of the I/O errors that I have observed in SAS were either related to the use of PROC SORT and could be resolved by the use of appropriate options for this procedure, or were related to the corruption of a dataset and is able to solve either by removing the corrupt dataset or repairing it (PROC DATASETS).

PROGRAMMING ADVICE

"*Datasets on a diet*" Don't forget that the size of the dataset file significantly impacts processing speed, so filter datasets before using them, keeping only the variables you need. Remember to declare variables, define an appropriate length, and save only the final dataset.

"*SAS raw dataset importation in the WORK library*" Make sure that you import all the raw data needed at the top of the program. You will avoid importing the same dataset two times (for distant data). If you are home-based worker like me, that can be an important timesaver.

"*Fast processing is not an end in itself*" Remember that choosing a technique you are not comfortable with just because it allows your program to run faster, increases the risk of missing an error and may complicate maintenance.

"*Decrease the data step number*" Sometimes use of procedures such as SQL or functions can help decrease the number of data steps in your program. It can also help shorten your programs and increase execution speed.

"What else?" When programming a lot of IF-THEN/ELSE statements, remember that the ELSE clause can make a big difference in performance. It is important to place the most expected conditions first and then eliminate the unnecessary IF statements, which can increase execution time.

CONCLUSIONS

How can programmers involved in optimizing drug development time help bring products to market earlier? First, programmers must be proactive and motivated to decrease development time. However, programmers also need to continue their training and need to foster mentoring relationships to acquire new skills. In addition, the availability of automated tools (macros, functions, etc.) allows programmers to concentrate on other tasks to increase productivity.

I am aware that some of these tips may seem incongruous: 'take the time to make a plan', 'taking the time to execute the step-by-step programs', 'schedule time to train'; indeed I am convinced that to be more effective we must take time to improve. But, I am also well aware that in the current performance of our duties and the many timelines that need to be met, it is not always easy or feasible to allocate time for training, etc. Finding a little time to improve your skills can save you much time later.

Finally, programmers should optimize the execution time of their programs. Indeed, a program that runs faster can be completed faster because programs in development often need to run numerous times before they are finished. Programmers must also be able to manage large datasets and to handle resource limitation problems, especially if working away from the office.

REFERENCES

[1] Ed Heaton, “Many-to-Many Merges in the DATA Step”, SAS Global Forum 2008. Available at: http://www2.sas.com/proceedings/forum2008/081-2008.pdf

[2] Gregg P. Snell, “Think FAST! Use Memory Tables (Hashing) for Faster Merging”, SUGI 31 Paper 244, Available at: http://www2.sas.com/proceedings/sugi31/244-31.pdf

[3] Leigh Ihnen and Mike Jones, “Improving SAS® I/O Throughput by Avoiding the Operating System File Cache” Available at https://support.sas.com/resources/papers/proceedings09/327-2009.pdf

ACKNOWLEDGMENTS

Once again I wish to thank Kurt without which my paper would have been a tissue of “frangliscisme”, dark and stodgy.

RECOMMENDED READING

SAS Tips and techniques

Helen Carey, Ginger Carey, “Tips and Techniques for the SAS® Programmer”, SAS Global Forum 2011 Paper 272, Available at: http://support.sas.com/resources/papers/proceedings11/272-2011.pdf

Kathy Harkins, Carolyn Maass, Mary Anne Rutkowski, “T.I.P.S. (Techniques and Information for Programming in SAS®)”, NESUG 2011 paper PO-088, Available at: http://analytics.ncsu.edu/sesug/2008/PO-088.pdf

Marje Fecht, “THINK Before You Type… Best Practices Learned the Hard Way”, SAS Global Forum 2009 Paper 133, Available at: http://support.sas.com/resources/papers/proceedings09/133-2009.pdf

SAS Function Compiler

Danny Rithy, “Getting Started with PROC FCMP”, WUSS 2015 Paper 54, Available at: http://www.lexjansen.com/wuss/2015/54_Final_Paper_PDF.pdf

Dylan Ellis, “RUN_MACRO Run! With PROC FCMP and the RUN_MACRO Function from SAS® 9.2, Your SAS® Programs Are All Grown Up”, SAS Global Forum 2013 Paper 033, Available at: http://support.sas.com/resources/papers/proceedings13/033-2013.pdf

Hashing

Mijun Hu, “It’s Not Just Fast Merging, It’s Powerful Hashing”, PharmaSUG China 2015 - Paper 14, Available at: http://www.lexjansen.com/pharmasug-cn/2015/PT/PharmaSUG-China-2015-PT14.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Mickael Borne
4Clinics
18 Rue Goubet
Paris / 75019
Work Phone: 01 84 19 35 59
Email: mborne@4clinics.com
Web: www.4clinics.com

APPENDIX I: USE OF FCMP PROCEDURE

```
/*************************************************************************
Platform: Windows
Version:  9.3
Author(s):Mickael Borne
Date:     02AUG2016
Purpose:  SAS FCMP example of use
          3 functions are created to perform the following action:
          round a figure when expected precision is defined in a variable of the
dataset
*************************************************************************/

proc fcmp outlib=WORK.MyFunCat.Example ;
*** Function setRoundingUnit: Return the rounding unit argument of the SAS round
function. *** ;
  function setRoundingUnit(ExpectedPrecision) ;
    length RoundingUnitC $ 15 ;
    *** RoundingUnit *** ;
    ** If ExpectedPrecision is: 0 or 1 2 3 Then RoundingUnit will be: 1.0 0.1 0.01
0.001... ** ;
    if (ExpectedPrecision<1) then Prec=0 ;
    else Prec=ExpectedPrecision ;
    select(Prec) ;
      when (0) RoundingUnit=1 ;
      otherwise do  ;
        RoundingUnitC='0.' ;
        do iter=1 to ExpectedPrecision ;
          if (iter=ExpectedPrecision) then do ;
            RoundingUnitC=compress(RoundingUnitC||'1') ;
            RoundingUnit=input(RoundingUnitC,best15.) ;
          end ;
          else RoundingUnitC=compress(RoundingUnitC||'0') ;
        end ;
      end ;
    end ;
    return( RoundingUnit ) ;
  endsub ;

  *** Function addTrailingZeros: Add if necessary trailing 0 to obtain a formatted
string. *** ;
  function addTrailingZeros(StringToTest $,DecimalePartExpectedLength) $ ;
    length TestedString $ 200 ;
    if (indexc(StringToTest,'.')<1) then DecimalePartObservedLength=0 ;
    else DecimalePartObservedLength=length(compress(scan(StringToTest,2,'.'))) ;
    if (DecimalePartObservedLength>DecimalePartExpectedLength) then do ;
      ** Case 1: ObservedLength>ExpectedLength then decimale part is truncated ** ;
      * This case is not expected * ;
TestedString=compress(scan(StringToTest,1,'.')||'.'||substr(scan(StringToTest,2,'.'),1
,DecimalePartExpectedLength)) ;
    end ;
    else if (DecimalePartObservedLength=DecimalePartExpectedLength) then do ;
      ** Case 2: ObservedLength=ExpectedLength then no transformation is done ** ;
      * Most frequent case nothing is done * ;
      TestedString=compress(StringToTest) ;
    end ;
    else if (DecimalePartObservedLength<DecimalePartExpectedLength) then do ;
      ** Case 3: ObservedLength<ExpectedLength then trailing zeros are added ** ;
      * Objectif of this function: manage this case * ;
      TestedString=compress(StringToTest) ;
      do while(DecimalePartObservedLength<DecimalePartExpectedLength) ;
        if (indexc(TestedString,'.')<1) then TestedString=compress(TestedString||'.0')
;
        else TestedString=compress(TestedString||'0') ;
```

```
        DecimalePartObservedLength=length(compress(scan(TestedString,2,'.'))) ;
      end ;
    end ;
    else do ;TestedString=compress(StringToTest) ;end ;
    return( TestedString ) ;
  endsub ;

  *** Function numToChar: Round and convert into formatted ASCII a number. *** ;
  function numToChar(NumberToFormat, Precision) $ ;
    length FormattedString $ 15 ;
    if (Precision<1) then Prec=0 ;
    else Prec=Precision ;
    if (NumberToFormat=.) then FormattedString='' ;
    else do ;

FormattedString=compress(put(round(NumberToFormat,setRoundingUnit(Prec)),best15.)) ;
      FormattedString=addTrailingZeros(FormattedString,Prec) ;
    end ;
    return( FormattedString ) ;
  endsub ;
run ;
** Created functions have to be accessible ** ;
options cmplib=WORK.MyFunCat ;

** Use of the setRoundingUnit function ** ;
data _NULL_ ;
  length ROUNDUNIT $ 15 ;
  do EXPECTEDPRECISION=0 to 5 ;
    ROUNDUNIT=setRoundingUnit(EXPECTEDPRECISION) ;
    put ROUNDUNIT ;
  end ;
run ;
** Use of the addTrailingZeros function ** ;
data _NULL_ ;
  length StringToTest $ 15 ;
  DecimalePartExpectedLength=5 ;
  do StringToTest='0.125', '0.58', '9.25897', '0.254789' ;
    RETURN=addTrailingZeros(StringToTest,DecimalePartExpectedLength)  ;
    put StringToTest= DecimalePartExpectedLength= RETURN= ;
  end ;
run ;
** Use of the addTrailingZeros function ** ;
data _NULL_ ;
  length RETURN $ 15 ;
  Precision=3 ;
  do NumberToFormat=0.125, 0.58, 9.25897, 0.254789 ;
    RETURN=numToChar(NumberToFormat, Precision)  ;
    put NumberToFormat= Precision= RETURN= ;
  end ;
run ;

data WORK.INPUTDATA ;
  length USUBJID $ 10 VISITID $ 3 PARAM $ 20 ;
  array parn {3} _temporary_ (1,2,3) ;
  array para {3} $20. _temporary_ ('XXX','YYY','ZZZ') ;
  array pard {3} _temporary_ (3,2,1) ;
  do PARCT=1 to dim(parn) ;
    PARAMN=parn[PARCT] ;
    PARAM=para[PARCT] ;
    PARDECNB=pard[PARCT] ;
    do VISITNUM=1, 2, 3, 4 ;
      VISITID=cats('V',put(VISITNUM,z2.)) ;
      do SUBJN=1 to 10 ;
        USUBJID=put(SUBJN, z10.) ;
```

```
        AVAL=ranuni(0) ;
        output ;
      end ;
    end ;
  end ;
run ;

** In one step using SQL procedure ** ;
proc sql ;
  create table WORK.DESCSTAT1 as
    select PARAMN, PARAM, PARDECNB, VISITNUM, VISITID, 1 as STATID, 'Mean (SD)' as
STATLAB length=20,
           catx(' ',numToChar(mean(AVAL), PARDECNB),cats('(',numToChar(std(AVAL),
PARDECNB+1),')')) as RESULT length=30
    from WORK.INPUTDATA group by PARAMN, PARAM, PARDECNB, VISITNUM, VISITID
    outer union corr
    select PARAMN, PARAM, PARDECNB, VISITNUM, VISITID, 2 as STATID, '[Min ; Max]' as
STATLAB,
cats('[',numToChar(min(AVAL), PARDECNB),';',numToChar(max(AVAL), PARDECNB),']') as
RESULT
    from WORK.INPUTDATA group by PARAMN, PARAM, PARDECNB, VISITNUM, VISITID
    order by PARAMN, VISITNUM, STATID ;
quit ;

** Or in two steps using STAT proc and DATA step ** ;
proc means data=WORK.INPUTDATA noprint ;
  by PARAMN PARAM PARDECNB VISITNUM VISITID ;
  output out=WORK.DESCSTAT0 mean=MEAN std=SD min=MIN max=MAX ;
  var AVAL ;
run ;

data WORK.DESCSTAT2 ;
  set WORK.DESCSTAT0 ;
  length STATLAB $ 20 RESULT $ 30 ;
  keep PARAMN PARAM PARDECNB VISITNUM VISITID STATID STATLAB RESULT ;
  STATID=1 ;
  STATLAB='Mean (SD)' ;
  RESULT=catx(' ',numToChar(MEAN, PARDECNB),cats('(',numToChar(SD, PARDECNB+1),')')) ;
  output ;
  STATID=2 ;
  STATLAB='[Min ; Max]' ;
  RESULT=cats('[',numToChar(MIN, PARDECNB),';',numToChar(MAX, PARDECNB),']') ;
  output ;
run ;
/*
proc sql ;
  create table WORK.VERIF as select l.PARAMN, l.VISITNUM, l.STATID, l.RESULT as RES1,
r.RESULT as RES2
    from WORK.DESCSTAT1 as l inner join WORK.DESCSTAT2 as r
    on l.PARAMN=r.PARAMN and l.VISITNUM=r.VISITNUM and l.STATID=r.STATID ;
  select * from WORK.VERIF where RES1 ne RES2 ;
quit ;
*/
```

APPENDIX II: INTRODUCTION TO HASH OBJECT

```
/*************************************************************************
Platform:               Windows
Version:                9.3
Author(s):              Mickael Borne
Date:                   02AUG2016
Purpose:                SAS HASH Object example of use
                        HASH object will be used to merge LB dataset with a parameter
dictionary (Many-To-One relationship)
*************************************************************************/
%put %sysfunc(getoption(WORK)) ;
** LB dataset creation ** ;
data WORK.LB ;
 *length STUDYID $ 15 DOMAIN $ 5 USUBJID $ 10 LBSEQ 8. LBTESTCD $ 8 LBTEST $ 40 LBCAT
LBORRES LBORRESU
         LBORNRLO LBORNRHI LBSTRESC $ 20 LBSTRESN 8. LBSTRESU $ 20 LBSTNRLO LBSTNRHI
8. LBNRIND LBSTAT $ 10
         LBREASND $ 200 LBNAM $ 70 LBSPEC $ 15 LBMETHOD $ 20 LBBLFL $ 1 LBFAST $ 1
VISITNUM 3. VISITID $ 3
         LBDTC $ 20 LBTPT $ 50 LBTPTNUM 8. ;
 length STUDYID DOMAIN USUBJID $ 200 LBSEQ 8. LBTESTCD LBTEST LBCAT LBORRES LBORRESU
LBORNRLO LBORNRHI LBSTRESC $ 200
        LBSTRESN 8. LBSTRESU $ 200 LBSTNRLO LBSTNRHI 8. LBNRIND LBSTAT LBREASND LBNAM
LBSPEC LBMETHOD LBBLFL LBFAST $ 200
        VISITNUM 3. VISITID LBDTC LBTPT $ 200
        LBTPTNUM 8. ;* in order to increase the dataset size character variables are
defined with a length of 200 * ;
  format PREVIOUSLBDT e8601dt19. ;
  call missing(LBSTAT,LBFAST,LBREASND,LBNAM) ;
  retain STUDYID 'STUDYXXX' DOMAIN 'LB' LBCAT 'CHEMISTRY' LBMETHOD 'BLOOD SAMPLING'
LBNRIND 'NORMAL' LBTPT '' LBTPTNUM 1 LBSEQ 0
         PREVIOUSLBDT . ;
  array parcd {5} $8. _temporary_ ('ALB','ALP','ALT','AST','CA') ;
  array par {5} $40. _temporary_ ('Albumin','Alkaline Phosphatase','Alanine Amino
Transferase','Aspartate Amino Transferase','Calcium') ;
  array paru {5} $20. _temporary_ ('g/dL','U/L','U/L','U/L','mmol/L') ;
  array parm {5} $20. _temporary_ ('','','','','') ;
  array spec {5} $20. _temporary_ ('SERUM','BLOOD','BLOOD','BLOOD','BLOOD') ;
  array nrlo {5} _temporary_ (3.4,44,10,10,2.125) ;
  array nrhi {5} _temporary_ (5.4,147,40,34,2.55) ;
  array nrby {5} _temporary_ (0.1,1,1,1,0.001) ;
  array DTSample {10} _temporary_
('01JAN14:09:18:00'dt,'03FEB14:08:35:00'dt,'12FEB14:10:26:00'dt,'15MAR14:09:24:00'dt,'
01APR14:07:48:00'dt,

'18JUN14:10:14:00'dt,'25JUN14:09:07:00'dt,'03JUL14:08:42:00'dt,'05JUL14:11:18:00'dt,'0
3AUG14:08:02:00'dt) ;
  do SUBJN=1 to 5000 ;
    USUBJID=put(SUBJN, z10.) ;
    LBSEQ=0 ;
    do VISITNUM=1 to 20 ;
      PARCT=0 ;
      VISITID=cats('V',put(VISITNUM,z2.)) ;
      if (VISITNUM=1) then do ;
        MINWGT=999 ;
        do VALUE=1 to 10 ;
          WEIGHT=ranuni(0) ;
          if (WEIGHT<MINWGT) then do ;
            LBDTC=put(DTSample[VALUE],e8601dt19.) ;
            MINWGT=WEIGHT ;
          end ;
```

```
        end ;
      end ;
      else if (PARCT=0) then LBDTC=put(intnx('DTMONTH',PREVIOUSLBDT,3,'S'),e8601dt19.)
;
      else ;
      do PARCT=1 to dim(parcd) ;
        LBTESTCD=parcd[PARCT] ;LBTEST=par[PARCT] ;
        LBORRESU=paru[PARCT] ;LBSTRESU=LBORRESU ;
        LBORNRLO=strip(put(nrlo[PARCT],best15.)) ;LBSTNRLO=nrlo[PARCT] ;
        LBORNRHI=strip(put(nrhi[PARCT],best15.)) ;LBSTNRHI=nrhi[PARCT] ;
        LBSPEC=spec[PARCT] ;
        ** A FCMP function could be useful for deriving the values of these parameters
** ;
        ** if we want to apply a model which is function of the parameter and the
visit for example ** ;
        ** but this is not the purpose of this example program ** ;
        MINWGT=999 ;
        do VALUE=nrlo[PARCT] to nrhi[PARCT] by nrby[PARCT] ;
          WEIGHT=ranuni(0) ;
          if (WEIGHT<MINWGT) then do ;
            LBSTRESN=VALUE ;
            MINWGT=WEIGHT ;
          end ;
        end ;
        LBORRES=compress(put(LBSTRESN,best15.)) ;
        LBSTRESC=LBORRES ;
        * In order to create a not sorted LB dataset we create randomly an ORDER
variable * ;
        ORDER=ranuni(0) ;
        LBSEQ+1 ;
        if (LBSEQ=1) then LBBLFL='Y' ;
        else LBBLFL='' ;
        output ;
      end ;
         PREVIOUSLBDT=input(LBDTC,e8601dt19.) ;
    end ;
  end ;
  drop PARCT MINWGT VALUE WEIGHT ;
run ;
/*
5000 Subjects, 20 Visits, 5 Parameters
NOTE: The data set WORK.LB has 500000 observations and 32 variables.
NOTE: DATA statement used (Total process time):
      real time           2:42.32
      cpu time            12.32 seconds
Dataset Size: 2 666 673 Ko
*/
data WORK.LBPARDICO ;
  length PARCAT1N 8. PARCAT1 $ 200 PARAMN 8. PARAMCD PARAM $ 200 PARDECNB PARCAT2N 8.
         PARCAT2 PARAMTYP LBCDLST CONVERSIONRULES BALLAST1-BALLAST20 LBTESTCD $ 200 ;
  call
missing(BALLAST1,BALLAST2,BALLAST3,BALLAST4,BALLAST5,BALLAST6,BALLAST7,BALLAST8,BALLAS
T9,BALLAST10,

BALLAST11,BALLAST12,BALLAST13,BALLAST14,BALLAST15,BALLAST16,BALLAST17,BALLAST18,BALLAS
T19,BALLAST20) ;
  array parn {5} _temporary_ (1,2,3,4,5) ;
  array parcd {5} $8. _temporary_ ('ALB','ALP','ALT','AST','CA') ;
  array par {5} $200. _temporary_ ('Albumin (g/dL)','Alkaline phosphatase
(U/L)','Alanine amino transferase (U/L)',
                                   'Aspartate amino transferase (U/L)','Calcium
(mmol/L)') ;
  array pardec {5} _temporary_ (1,0,0,0,3) ;
```

```
  array parcdl {5} $200. _temporary_ ('ALB, ALBUMIN','ALP, AP','ALT','AST','CA,
CALCIUM') ;
  array conv {5} $200. _temporary_ ('','','','','1mg/dL = 0.25mmol/L') ;
  retain PARCAT1N 1 PARCAT1 'Clinical chemistry tests' PARCAT2N 1 PARCAT2 'Laboratory
safety'
         PARAMTYP 'Original laboratory data' ;
  do PARCT=1 to dim(parcd) ;
    PARAMN=parn[PARCT] ;PARAMCD=parcd[PARCT] ;PARAM=par[PARCT] ;
    PARDECNB=pardec[PARCT] ;LBCDLST=parcdl[PARCT] ;CONVERSIONRULES=conv[PARCT] ;
    do i=1 to countw(LBCDLST) ;
      LBTESTCD=scan(LBCDLST,i) ;
      output ;
    end ;
  end ;
  do PARCT=6 to 2000 ;
    PARAMN=PARCT ;PARAMCD=cats('PAR',put(PARCT,best5.)) ;
    PARAM=catx(' ','Label for parameter',PARAMCD) ;
    PARDECNB=0 ;LBCDLST=PARAMCD ;CONVERSIONRULES='' ;
    LBTESTCD=PARAMCD ;
    output ;
  end ;
  drop PARCT i ;
run ;

*** Join LB and LBDICO using the HASH object *** ;
data WORK.JOIN1 ;
  set WORK.LB ;
  length PARAMN 8. PARAMCD $ 8 PARAM $ 40 PARDECNB 8. ;* Variables comming from HASH
object have to be declared * ;
  call missing(PARAMN,PARAMCD,PARAM,PARDECNB) ;* Used to avoid Uninitialized notes in
the log file * ;
  if (_N_=1) then do ;
    declare hash dico(dataset:"WORK.LBPARDICO") ;* Hash object declaration * ;
      dico.defineKey('LBTESTCD') ;* Key definition * ;
      dico.defineData('PARAMN','PARAMCD','PARAM','PARDECNB') ; * Accessible data * ;
      dico.defineDone() ;
  end ;
  rc=dico.find(key:LBTESTCD) ;* join * ;
  if (not rc) then do ;* LBTESTCD found in the dictionary * ;
    *** Character conversion of the numerical results (AVAL to AVALC) *** ;
    ** Use of PARDECNB to round results ** ;
    *AVALC=compress(numToChar(AVAL, max(0,PARDECNB))) ;
    output ;
  end ;
  else ;* LBTESTCD not found in the dictionary * ;
  drop rc ;
run ;
/*
NOTE: DATA statement used (Total process time):
      real time           2:15.54
      cpu time            11.12 seconds
*/

*** Creation of the Not sorted LB dataset *** ;
proc sort data=WORK.LB out=WORK.LBNOTSORTED ;
  by ORDER ;
run ;
/*
NOTE: There were 500000 observations read from the data set WORK.LB.
NOTE: The data set WORK.LBNOTSORTED has 500000 observations and 32 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           17:51.52
      cpu time            18.48 seconds
*/
** Another way to do the same ** ;
```

```
proc sql ;
  create table WORK.LBNOTSORTED2 as select * from WORK.LB order by ranuni(0) ;
quit ;
/*
NOTE: The query as specified involves ordering by an item that doesn't appear in its
SELECT
      clause.
NOTE: Table WORK.LBNOTSORTED2 created, with 500000 rows and 32 columns.
97   quit ;
NOTE: PROCEDURE SQL used (Total process time):
      real time           8:17.35
      cpu time            21.49 seconds
*/
** Optimization ** ;
proc sort data=WORK.LB out=WORK.LBNOTSORTED3 noequals ;
  by ORDER ;
run ;
/*
NOTE: The data set WORK.LBNOTSORTED3 has 500000 observations and 32 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           5:50.83
      cpu time            15.64 seconds
*/
*** Creation of the Not sorted dictionary *** ;
proc sql ;
  create table WORK.LBPDNOTSORTED as select * from WORK.LBPARDICO order by ranuni(0) ;
quit ;

*** Join LB and LBDICO using the classic MERGE statement *** ;
proc sort data=WORK.LBNOTSORTED out=WORK.LBSORTED ;
  by LBTESTCD ;
run ;
proc sort data=WORK.LBPDNOTSORTED out=WORK.LBPDSORTED ;
  by LBTESTCD ;
run ;
data WORK.JOIN2 ;
  merge WORK.LBSORTED(in=a) WORK.LBPDSORTED(keep=LBTESTCD PARAMN PARAMCD PARAM
PARDECNB) ;
  by LBTESTCD ;
  if (a) ;
run ;
/*
NOTE: PROCEDURE SORT used (Total process time):
      real time           2:47.40
      cpu time            20.20 seconds
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.62 seconds
      cpu time            0.01 seconds
NOTE: DATA statement used (Total process time):
      real time           1:36.99
      cpu time            11.40 seconds
*/
*** Join LB and LBDICO using the PROC SQL *** ;
proc sql ;
  create table WORK.JOIN3 as
    select l.*, r.PARAMN, r.PARAMCD, r.PARAM, r.PARDECNB
    from WORK.LBNOTSORTED as l left join WORK.LBPDNOTSORTED as r
    on l.LBTESTCD=r.LBTESTCD ;
quit ;
/*
NOTE: PROCEDURE SQL used (Total process time):
      real time           3:58.46
      cpu time            35.49 seconds
*/
```