PhUSE 2011

Paper CS05

SAS Goes Spreadsheet

Raimund Storb, Boehringer Ingelheim Pharma KG

ABSTRACT

Since Dan Bricklin and Bob Frankston implemented the first WYSIWIG spreadsheet program with VisiCalc® in 1979, spreadsheets have become a common tool for End-user development. Programmers face the challenges of implementing algorithms defined in spreadsheets and handling problems defined within them.
One of the major issues programmers face is that both the data and the algorithms in a spreadsheet are not sequentially defined. Also the definition within a spreadsheet may apply to special parts, like different BY groups, within a SAS(R) data set.
One possible solution to these issues might be to organize the data in a DoW-Loop (Whitlock DO Loop) and define the algorithm within this structure.
This paper explores how to define certain operations within a spreadsheet, for example how to copy a formula or how to define the dimension of the array(s) defining the spreadsheet and which of these tasks can be predefined with a SAS® macro. The paper will also describe some of the common mistakes and pitfalls of using this approach for programming.

INTRODUCTION

Since people start working with ledger sheets or other two dimensional organized data on a piece of paper it became common to process and display data in this form. As a consequence of this people invented programs which process (and display) data in that way in a computer application.
With the development of office suits this applications became commonly used.
The main concept of these spreadsheets is the grid of cells to organize and display the data. Another aspect is the formulas that are mostly also organized in the same electronic document.

In opposite relational databases commonly uses only one observation (one row) at a time. This is quite efficient when you have all data you need to process in a single observation or even in some single observation of different tables. They can be easily merged or joined. That's what these systems provide.

With a two dimensional array you may organize the data within a data step to organize data from more than one observation in a grid for random access.

Defining arrays to keep the needed information from the by groups is often called a DOW loop, a Do Loop of Whitlock or a Dorfman Whitlock loop in respect to the programmer first described this technique (Paul Dorfman) and the other programmer promoting this (Ian Whitlock).

This paper shows how to handle this structure and will provide some examples.

To read further you best remind the following.

1) An array, in any programming language, also a multidimensional array, is a piece of continuous memory for random access via indices(s). The default behavior for arrays in SAS® is like storing pointer to variables within the arrays and handle access to a piece of the array as an access to those variables referenced by the pointer. Defining the array as _temporary_ changes this.
2) Arrays in SAS® didn't change there values when a new observation is read. But the variables they referring to may do.
3) You can only have one type of data in an array, numeric or character variables.
4) If you define a temporary array of character variables you need to define the proper length for these variables.

5) A set statement in a data step controls the input vector which contains all values of the observation read from a single observation. When this set statement is controlled in an explicit loop and not in the implicit data step loop then the subsequent statements are as often repeated as the loops are processed.
6) An output statement in a data step controls the analogue for the output. The created data set will contain as many observations as the according output statement is processed.
7) A SAS® format may give you the possibility to look up values within a data step. For example the index in an array representing the values contained in a variable.

HOW TO SET UP YOUR ARRAY AND HOW TO WRITE THE RESULTS.

In this chapter we will see how to setup the data structure of a DOW loop in a standardized way by applying macros.

The main tasks here are

- Defining the array.
- Filling the data in the array
- Write out the data

DEFINIG THE ARRAY

First of all we need the dimensions of the arrays. One dimension is defined by the number of variables you want to use. The other dimension is defined by the maximum number of observations you want to use information, most likely the maximum number of observations in a by group.
For the first dimension you have to consider what you want to program, the second can be easily determined with the proc sql procedure. But the second dimension is determined by the data, so it may change and should be stored in a macro variable.

Second you need to determine which dimension of the arrays is representing the variables; the other will represent the observation. You also need to decide which variable is represented with which index.
In this paper the first dimension will always represent the variables. All variables will be represented in the arrays. This information can be obtained with the following macros.

The first macro counts the maximum number of observations in the largest by group.

```
%macro count_row( _ds = , _vars = , _result = , _rc = rc );
************************************************;
* Macro to calculate the max. rows for a by group *;
* _ds   : Name of the dataset                     *;
* _vars : Variable names defining the by group    *;
* _result : Name of macro variable containing the *;
*           result                                *;
************************************************;

  %local list ;
  %global &_result. &_rc.;

  data _null_;
    list = tranwrd( strip( "&_vars." ) ,' ',',');
    call symput('list',list);
  run;
  %let list = &list.;
  %put Counting max. rows for by groups &list.;

  proc sql noprint;
    select max(count) into : &_result
    from   ( select count(*) as count
             from   &_ds.
             group  by &list. );
  quit;
  %let &_result. = &&&_result..;

  %put Max. rows for by groups &list. is &&&_result..;
```

```
  %let &_rc.=&sqlrc.;

%mend  count_row;
```

We may decide to enlarge the array for additional variables you want to define. We have to keep in mind that the simple new defined variable will be defined for whole by groups. If a certain date has to be calculated for any observation we read from the input data set we need an extra column in one of the arrays defined.

The second macro obtaining two lists of variables and the maximum length of character variables. If wanted it also writes two formats giving you the position of a variable in this lists.

```
%macro get_cols( _ds = , c_list = , c_len = , n_list = , _rc = rc , _writef = Y);
************************************************;
* Obtain list of variables from dataset _DS      *;
* _ds   : Name of the dataset                    *;
* c_list: List of character variables            *;
* n_list: List of numeric   variables            *;
* c_len : Max length of character variables      *;
************************************************;
  %let &_rc. = 0;
  %global &c_list. &c_len. &n_list.;
  ods output variables=_as_is ( keep = variable type len );

  proc contents data = &_ds.;
  quit;

  ods output close;

  *** Sort data by variable names to be indepentent from the ***;
  *** Dataset internal order f representation               ***;
  proc sort data = _as_is;
    by type variable;
  run;

  proc sql noprint;
    select max(len) into : &c_len
    from   _as_is
    where  type = 'Char';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
    select strip(variable) into : &c_list
    separated by ' '
    from   _as_is
    where  type = 'Char';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
    select strip(variable) into : &n_list
    separated by ' '
    from   _as_is
    where  type = 'Num';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
  quit;

  %let &c_len. = &&&c_len..;
  %put Max. length of character variables (&c_len.) is &&&c_len..;
  %put List of character variables (&c_list.): &&&c_list.;
  %put List of numeric variables   (&n_list.): &&&n_list.;
  %if &_writef = Y and &&&_rc.. = 0 %then %do;
      data _cntlC;
        retain fmtname 'varC2n' type 'I';
        set _as_is ( where = ( _t = 'Char' ) rename = ( type = _t) );
        start = variable;
        label = put(_N_,3.);
      run;
      proc format cntlin = _cntlc;
```

```
      run;

      data _cntlN;
        retain fmtname 'varN2n' type 'I';
        set _as_is ( where = ( _t = 'Num' ) rename = ( type = _t) );
        start = variable;
        label = put(_N_,3.);
      run;
      proc format cntlin = _cntln;
      run;
  %end;

%mend  get_cols;
```

FILLING THE DATA IN THE ARRAY

To fill the array for a by group means to loop through the by group and storing the data in the array. Please note that you have to initialize the cells of the array that you don't fill in or you have to consider that these cells are containing the values from the by group before. For this you should count the current observation number in the by group. Most simply you can assign the values to the temporary arrays by using arrays tied to the input variables within a simple loop. Assuming that the dimensions and lists of variables are stored in macro variables a piece of code (in a macro) implementing this may look like this.

```
data two;
  *** Define arrays and tie them to the input variables **
  array c_in [ &c_col_cnt. ] $&_clength. &_ccols.;
  array n_in [ &n_col_cnt. ] 8. &_ncols.;
  *** Define the arrays for the dow loop ***;
  array _c [ &_nrows. , &c_all_cnt. ] $&_clength. _temporary_;
  array _n  [ &_nrows. , &n_all_cnt. ] 8.  _temporary_;
  *** Reading the data ***;
  do until( last.var_n or fin );
    set one end = fin ;
    by  var1 var2 … var_n;
    *** Initialise a counter for the obs / rows read in ***;
    if first.var_n then _row = 1;
                   else _row = _row + 1;
    do i = 1 to &c_col_cnt. ;
       _c[ _row , i ] = c_in[ i ];
    end;
    do j = 1 to &n_col_cnt. ;
       _n[ _row , j ] = n_in[ j ];
    end;
  end;
```

Please note that in fact there are only as many pass through the subsequent code as there are by groups in the data set. But the hole data of a by groups is stored in the arrays for random access any time the subsequent code is processed.
Also note that the number of observations of the last processed by group is stored in a variable. We'll need this value for controlling the access to the data in the array. Not to avoid any subscript out of range errors but to avoid accessing data that is not defined for the current by group.

WRITE OUT THE DATA

Processing the data in the fanciest way is pointless if we can't store the results.
Therefore, before describing how to work with the data stored in this array let's have a look on the possibility to write out all the data in a data step.
Without any appropriate code we would only keep the input data from the last observation of any by group. As mentioned before, subsequent code after reading in the data set in a loop is only processed once for any by group. Thus we need to decide how to store the data for a by group.

When storing the data in a data set we can use the output statement in a similar way we used the set statement before whilst reading in the data.
The following piece of code may be used to put out all the values stored in the array in the same order the values have been stored including all changes. A prerequisite would be to tie the output variables to two arrays called C_out and n_out.

```
if last.var_n then do;
  do k = 1 to _row;
    do i = 1 to &c_all_cnt. ;
      c_out[ i ] = _c[ k , i ] ;
    end;
    do j = 1 to &n_all_cnt. ;
      n_out[ j ] = _n[ k , j ] ;
    end;
    output;
  end;
  _row = .;
end;
```

When needed we may change the way of putting out the data to transpose data or we may only store chosen values from the arrays.

HOW TO HANDLE THE ARRAY IN BETWEEN READING IN AND WRITING OUT DATA

After discussing how to read in and write out the data we will discuss the main task. Processing the data between these steps. First we will define some simple macros to help with this task.

CALLING DATA BY NAME

The native way to address values in an array is via the numeric indices. A common problem here would be the insertion of a column (here a variable in the input data set) that shows up before the indices when use. When doing so, all the references after the inserted column number won't work any more in a proper way. That's why a way to address the according index by variables name is wanted. And that's also the reason why the second macro shown above to help defining these Arrays has an option for creating two formats which gives access to the indices via variable names.
The following two macros illustrates how to use these formats.

```
%macro getnum( i , _var );
  _n [ &i. , input( "&_var." , $VARN2N. ) ]
%mend getnum;

%macro gettxt( i ,_var );
  _c [ &i. , input( "&_var." , $VARC2N. ) ]
%mend gettxt;
```

In deed both macros doesn't do more than giving us a reference to an array cell. We may use this to read data from, or to write data to this cell.

HOW TO RUN SIMPLE STATISTICAL FUNCTIONS ON THIS DATA STRUCTURE.

The SAS System has a variety of statistical functions like minimum or maximum. One of this function has a difference in use which has to be considered later. All functions with exception of the nmiss function handle missing values in the way that this value is ignored. But the nmiss function counts them

To evaluate such a function on a continuous piece of a row we simply have to loop over the continuum. This can be easily handled with the macro below. Here the the variable _stat would determine the function, _start and _end the first and last row and _var would contain the name of the variable.

```
%macro _stat_function ( _stat = , _start = , _end =  , _var = );
  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min, mad etc.              ***
  *** _start and _end the start and end row to evaluate   ***
  *** _ var should be the name associated to the column   **/
  %local __list __var;
  %let   __var = %sysfunc(upcase(&_var));
  %do i = &_start %to &_end. - 1;
```

```
    %let __list = &__list. _n [ &i.  , input( "&__var." , VARN2N. ) ] , ;
  %end;
  %let __list = &__list.  _n [ &_end. , input( "&__var." , VARN2N. ) ] ;
  &_stat. ( &__list. ) ;
%mend  _stat_function ;
```

The second last line is needed because the list of arguments shouldn't end with a colon. A different approach is simply to loop over all rows and insert a dot at the end. This missing value is ignored by all but not the nmiss function.

But some times we may want to use a function not only on one column. How to do so by defining the first and the first and the last column is shown below.

```
%macro _stat_function2 ( _stat = , _start = , _end = , _var1 = , _var2 = );

  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min etc.                    ***
  *** _start and _end the start and end row to evaluate    ***
  *** _var1/1 should be the name associated to the         ***
  *** first / last column                                  **/

  %local __list __var1  __var2 i j ;
  %let   __var1 = %sysfunc(upcase(&_var1));
  %let   __var2 = %sysfunc(upcase(&_var2));

  %do i = &_start. %to &_end. 1;
      %do j = %sysfunc(input( "&__var1." , VARN2N. ))
          %to %sysfunc(input( "&__var2." , VARN2N. ))
            %if &i = &end and &j = %sysfunc(input( "&__var2." , VARN2N. ))
            %then %let __list = &__list. _n [ &i.  , &j. ] ;
            %else %let __list = &__list. _n [ &i.  , &j. ] , ;
      %end;
  %end;

  &_stat. ( &__list. ) ;
%mend  _stat_function2 ;
```

Be aware that here the sequence how the variable are defined in the array are used to define which vales will be used. If the macro given in the chapter before is used, any change of the program before may change this sequence.

More over we may want to use values not stored in sequential columns. For this we may want to apply a list of variable names to the macro. And once more the code is below.

```
%macro _stat_function_list ( _stat = , _start = , _end = , _varlist = examdt ) ;
  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min etc.                    ***
  *** _start and _end the start and end row to evaluate    ***
  *** _varlist should be the names associated to the       ***
  *** columns wanted                                       **/

  %local __list __num __i __j __var ;
  %let   __vlist = %sysfunc(upcase(&_varlist.));
  %let   __vlist = %sysfunc(strip(&__vlist.));
  %let   __num   = %sysfunc(countw(&__vlist.));

  %do __I = 1 %to &__num.;
      %let __var = %qscan( &__vlist. , &__i. );
      %do __j = &_start. %to &_end.;
        %if &__I = &__num and &__j = &_end.
         %then %let __list = &__list. _n [ &__j.  , input( "&__var." , VARN2N. ) ] ;
         %else %let __list = &__list. _n [ &__j.  , input( "&__var." , VARN2N. ) ] ,
;
```

```
      %end;
  %end;

  &_stat. ( &__list. ) ;
%mend  _stat_function_list ;
```

A SIMPLE EXAMPLE FROM LABORATORY DATA

Sometimes we have do derive values from laboratory data. Some of the laboratory data is derived. The values we will focus on are absolute neutrophile count (ANC), neutrophiles(absolute and relative) (NEUTAB,NEUT), bands (BANDS) and eosinophiles (EOS). The absolute neutrophile count can be calculated as the sum from absolute netrophiles and bands. The relative neutrophiles are calculated relative to eosinophile,

The task was to calculate ANC where is was missing.

SIMPLE SOLUTION

When ever needed calculate ANC as BANDS plus NEUTAB. If NEUTAB is missing and needed to calculate ANC calculate it from NEUT and EOS. If BANDS is missing then take NEUTAB as ANC. I In deed the bands are the few from ANC which are not counted to the neutrophiles. So we may standardize the values and then transpose the data based on study subject and time point. This works fine when there is at maximum one observation per laboratory parameter on this key.

LESS SIMPLE SOLUTION

Taking into account that the data is not unique per study, subject and time point we need to ensure that the given BANDS and NEUTAB values we want to use are from the same sample. For example having the ANC value for one sample may show which BANDS and NEUTAB value belongs to that sample. Thus the other will belong to the sample with missing ANC. Here we have clearly an advantage by having all relevant data in random access in an array. More over we should be able to implement a rule a physician can show us when presented the data in an spreadsheet.

AN EXAMPLE FROM RECIST DATA

RECIST is based on three sorts of data used for an accessment. Data about target leasons, non target leasons and new leasons. The data from an accessment is sent to an expert for an opinion. All leasons in an accessment may have a slightly different examination date

The task was to find wrong accessment identifier, wrong dates and missing values.

SOLUTION

As a first step all missing examination dates, study days, assessment identifier, measurements, missing status for non target leasons were marked in an extra column.

In a second step Search for an observation with missing image or status and assessmenat identifier but date like the corrected one, search also for an observation which fits to the assessmenat identifier for that leason.
If both are present then the missing image is probably repoted to the wrong cycle. Store the row number to these observations in an extra column.

In a last step mark those observations which are probably obsolete, have a wrong cycle etc. Use the reference in the array to gather the items needed to write a description of the detected issue like it is shown in the picture below.

	Date of imaging	IMAGE	Imaging technique	Lesion number	Lesion measuremen	Patient number	TRTDAY	Treatment course	THISMISS	MISSTYPE
414	.	No	.	Lesion no 6	.	391065	.	6	Probably the cycle for a wrong reported image	Check with the observation for leason Lesion no 6 with cycle 4 or date 14MAR11
415	24SEP2010	Yes	CT spiral	Lesion no 1	15.00	391065	-4	0		
416	24SEP2010	Yes	CT spiral	Lesion no 2	14.00	391065	-4	0		
417	24SEP2010	Yes	CT spiral	Lesion no 3	12.00	391065	-4	0		
418	24SEP2010	Yes	CT spiral	Lesion no 4	12.00	391065	-4	0		
419	24SEP2010	Yes	CT spiral	Lesion no 5	13.00	391065	-4	0		
420	24SEP2010	Yes	CT spiral	Lesion no 6	26.00	391065	-4	0		
421	24SEP2010	Yes	CT spiral	Lesion no 7	17.00	391065	-4	0		
443	17JAN2011	Yes	CT spiral	Lesion no 1	7.00	391065	112	4		
444	17JAN2011	Yes	CT spiral	Lesion no 2	12.00	391065	112	4		
445	17JAN2011	Yes	CT spiral	Lesion no 3	9.00	391065	112	4		
446	17JAN2011	Yes	CT spiral	Lesion no 4	7.00	391065	112	4		
447	17JAN2011	Yes	CT spiral	Lesion no 5	11.00	391065	112	4		
448	17JAN2011	Yes	CT spiral	Lesion no 6	25.00	391065	112	4	Probably the right image for this cycle	Check with the observation for leason Lesion no 6 with cycle 4 or date 14MAR11
449	17JAN2011	Yes	CT spiral	Lesion no 7	17.00	391065	112	4		
450	14MAR2011	Yes	CT spiral	Lesion no 1	7.00	391065	168	6		
451	14MAR2011	Yes	CT spiral	Lesion no 2	11.00	391065	168	6		
452	14MAR2011	Yes	CT spiral	Lesion no 3	10.00	391065	168	6		
453	14MAR2011	Yes	CT spiral	Lesion no 4	10.00	391065	168	6		
454	14MAR2011	Yes	CT spiral	Lesion no 5	13.00	391065	168	6		
455	14MAR2011	Yes	CT spiral	Lesion no 6	24.00	391065	168	4	Probably a wrong cycle reported	Check with other observations for leason Lesion no 6 with cycle 6 and date . and also with cycle 4 and date 17JAN11
456	14MAR2011	Yes	CT spiral	Lesion no 7	14.00	391065	168	6		

CONCLUSION

The introduced techniques and macros are not simple but not too complicate to use. And in some cases they are easier and much better to use then to ignore.

How ever, I want to invite every body to “play” a little bit with this. At minimum this will lead to a better understanding of what’s happen when an array is used in a data step and how implicit and explicit loops work in a data step.

There fore the macros are added to the appendix. This code also includes some example calls. These examples use a copy of the sashelp.heart data set.

REFERENCES

Richard Read Allen: Practical Uses of the DOW Loop in Pharmaceutical Programming. PhUse 2009, Paper Tu01
http://www.lexjansen.com/phuse/2009/tu/tu01.pdf
Paul Dorfman: The DOW-Loop Unrolled
http://analytics.ncsu.edu/sesug/2010/BB13.Dorfman.pdf

ACKNOWLEDGMENTS

I would like to thank Peter Craffort for insights in the processing of a simple data step and the use of arrays he shows me a decade ago. I would also like to thank James Witt for some discussions about DOW loops 2 years ago. Without this both this paper wouldn’t exist.

APPENDIX

```
/*
data _tst;
  set sashelp.heart;
run;
/*  */

%macro count_row( _ds = , _vars = , _result = , _rc = rc );
************************************************;
* Macro to calculate the max. rows for a by group *;
* _ds   : Name of the dataset                     *;
* _vars : Variable names defining the by group    *;
```

```
* _result : Name of macro variable containing the *;
*           result                                 *;
************************************************;

  %local list ;
  %global &_result. &_rc.;

  data _null_;
    list = tranwrd( strip( "&_vars." ) ,' ',',');
    call symput('list',list);
  run;
  %let list = &list.;
  %put Counting max. rows for by groups &list.;

  proc sql noprint;
    select max(count) into : &_result
    from   ( select count(*) as count
             from   &_ds.
             group  by &list. );
  quit;
  %let &_result. = &&&_result..;

  %put Max. rows for by groups &list. is &&&_result..;
  %let &_rc.=&sqlrc.;

%mend  count_row;

/*
%global rc r;
%count_row( _ds     = _tst
          , _vars   = SMOKING_STATUS WEIGHT_STATUS BP_STATUS AGEATSTART
          , _result = r );
%put &rc. &r.;
/*  */

%macro c_length( _ds = , _vars = , _result = , _rc = rc );
  %local list ;

  data _null_;
    list = tranwrd( strip( "&_vars." ) ,' ',"),length(");
    call symput('list','max(length(' !! list !! '))' );
  run;
  %let list = &list.;
  proc sql noprint;
    create table __cnt as
    select &list. as cnt, 1 as grp
    from   &_ds;
    select max(cnt) into : &_result
    from   __cnt
    group  by grp;
  quit;
  %let &_result. = &&&_result..;
  %put Max. length of character variable is &&&_result..;
  %let &_rc.=&sqlrc.;

%mend  c_length;

/*
%global rc char_l;
%c_length( _ds     = _tst
          , _vars   = SMOKING_STATUS WEIGHT_STATUS BP_STATUS
          , _result = char_l );
%put &rc. &char_l.;
```

```
/*  */
%macro get_cols( _ds = , c_list = , c_len = , n_list = , _rc = rc , _writef =
Y);
  %let &_rc. = 0;
  %global &c_list. &c_len. &n_list.;
  ods output variables=_as_is ( keep = variable type len );

  proc contents data = &_ds.;
  quit;

  proc sql noprint;
    select max(len) into : &c_len
    from   _as_is
    where  type = 'Char';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
    select strip(variable) into : &c_list
    separated by ' '
    from   _as_is
    where  type = 'Char';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
    select strip(variable) into : &n_list
    separated by ' '
    from   _as_is
    where  type = 'Num';
    %let &_rc.=%sysfunc(max( &&&_rc.. , &sqlrc. ));
  quit;

  %let &c_len. = &&&c_len..;
  %put Max. length of character variables (&c_len.) is &&&c_len..;
  %put List of character variables (&c_list.): &&&c_list.;
  %put List of numeric variables   (&n_list.): &&&n_list.;
  %if &_writef = Y and &&&_rc.. = 0 %then %do;
      data _cntlC;
        retain fmtname 'varC2n' type 'I';
        set _as_is ( where = ( _t = 'Char' ) rename = ( type = _t) );
        start = variable;
        label = put(_N_,3.);
      run;
      proc format cntlin = _cntlc;
      run;

      data _cntlN;
        retain fmtname 'varN2n' type 'I';
        set _as_is ( where = ( _t = 'Num' ) rename = ( type = _t) );
        start = variable;
        label = put(_N_,3.);
      run;
      proc format cntlin = _cntln;
      run;

      data _cntl;
        set _cntlc _cntln;
        type = 'C';
      run;
      proc format cntlin = _cntl;
      run;
  %end;

%mend  get_cols;

/*
%global c_cols n_cols char_len;
%get_cols( _ds = _tst
```

```
         , c_list = c_cols
         , c_len =  char_len
         , n_list = n_cols
         , _rc = rc );
%put &c_cols &n_cols &char_len;
/*  */

%macro def_loop( _dsin= , _dsdef= , _by=
               , _nrows= , _ncols= , _ccols= , _c_new= , _n_new=
               , _c_cnt= , _n_cnt=
               , _clength= , _rc = rc );
  %local c_col_cnt n_col_cnt last_by;
  %global &_c_cnt. &_n_cnt.;

  %let &_rc=0;
  %let c_col_cnt = %sysfunc(countw( &_ccols. ));
  %let n_col_cnt = %sysfunc(countw( &_ncols. ));
  %let c_all_cnt = %sysfunc(countw( &_ccols. &_c_new. ));
  %let &_c_cnt = &c_all_cnt. ;
  %let n_all_cnt = %sysfunc(countw( &_ncols. &_n_new. ));
  %let &_n_cnt = &n_all_cnt. ;
  %let last_by   = %sysfunc(scan( &_by , -1 ));
  %put Number of num. - alphanum. variables : &n_col_cnt. - &c_col_cnt. ;
  %put Max number of rows: &_nrows. - last by variable: &last_by. ;

  proc sort data = &_dsin;
    by &_by.;
  run;
  %let &_rc.=%sysfunc(max( &&&_rc. , &sysrc. ));

  data &_dsdef. ;
    array _c [ &_nrows. , &c_all_cnt. ] $&_clength. _temporary_;
    array _n  [ &_nrows. , &n_all_cnt. ] 8.  _temporary_;
    array c_in [ &c_col_cnt. ] $&_clength. &_ccols.;
    array n_in [ &n_col_cnt. ] 8. &_ncols.;
    array c_out [ &c_all_cnt. ] $&_clength. &_ccols. &_c_new.;
    array n_out [ &n_all_cnt. ] 8.          &_ncols. &_n_new.;
    retain _row .;
  do until( last.&last_by. or fin );
    set &_dsin. end = fin ;
    by  &_by.;
    if first.&last_by. then _row = 1;
                       else _row = _row + 1;
    do i = 1 to &c_col_cnt. ;
       _c[ _row , i ] = c_in[ i ];
    end;
    do j = 1 to &n_col_cnt. ;
       _n[ _row , j ] = n_in[ j ];
    end;
  end;

%mend  def_loop;

/* This code demonstrate that the subsequent code is processed once per by
group.
%def_loop( _dsin=  _tst
         , _dsdef= _tst2
         , _by=  SMOKING_STATUS WEIGHT_STATUS BP_STATUS AGEATSTART
         , _nrows= &r.
         , _ncols= &n_cols.
         , _ccols= &c_cols.
         , _c_new= c_test
         , _n_new= n_test
```

```
         , _c_cnt= all_char
         , _n_cnt= all_num
         , _clength=&char_len. );
run;
/*  */

%macro end_loop( c_all_cnt= , _by= , n_all_cnt= , _rc = rc );
  %local last_by;
  %let last_by   = %sysfunc(scan( &_by , -1 ));

    if last.&last_by. then do;
      do k = 1 to _row;
        do i = 1 to &c_all_cnt. ;
          c_out[ i ]  = _c[ k , i ] ;
          _c[ k , i ] = ' ' ;
        end;
        do j = 1 to &n_all_cnt. ;
          n_out[ j ]  = _n[ k , j ] ;
          _n[ k , j ] = . ;
        end;
        output;
      end;
      _row = .;
    end;
  run;

  %let &_rc=0;
  %let &_rc.=%sysfunc(max( &&&_rc. , &sysrc. ));
%mend  end_loop;

/* Compare the number of observations of teh result with the example before
 I If you want to play just change the _by option in the second macro call to
BP_STATUS
%global all_char all_num;

%def_loop( _dsin=  _tst
         , _dsdef= _tst2
         , _by=  SMOKING_STATUS WEIGHT_STATUS BP_STATUS AGEATSTART
         , _nrows= &r.
         , _ncols= &n_cols.
         , _ccols= &c_cols.
         , _c_new= c_test
         , _n_new= n_test
         , _c_cnt= all_char
         , _n_cnt= all_num
         , _clength=&char_len. );

    n_test = _row;
    c_test = put(_row , best. ) !! ' test';

%end_loop( c_all_cnt= &all_char.
         , n_all_cnt= &all_num.
         , _by = AGEATSTART
         , _rc = rc );
/*  */

%macro getnum( i , _var );
  _n [ &i. , input( "&_var." , VARN2N. ) ]
%mend getnum;


%macro gettxt( i ,_var );
  _c [ &i. , input( "&_var." , VARC2N. ) ]
```

```
%mend gettxt;

%macro Ccompare( i1 , _var1 , i2 , _var2 );
   _c [ &i1. , input( "&_var1." , VARC2N. ) ] = _c [ &i2. , input( "&_var2." ,
VARC2N. ) ]
%mend  Ccompare;

%macro Ncompare( i1 , _var1 , i2 , _var2 );
   _N [ &i1. , input( "&_var1." , VARN2N. ) ] = _N [ &i2. , input( "&_var2." ,
VARN2N. ) ]
%mend  Ncompare;



%macro _stat_function ( _stat = , _start = , _end =  , _var = );
  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min etc.                    ***
  *** _start and _end the start and end row to evaluate    ***
  *** _var should be the name associated to the column     **/

  %local __list __var;
  %let   __var = %sysfunc(upcase(&_var));
  %do i = &_start %to &_end. - 1;
    %let __list = &__list. _n [ &i.  , input( "&__var." , VARN2N. ) ] , ;
  %end;
  %let __list = &__list.  _n [ &_end. , input( "&__var." , VARN2N. ) ] ;
  &_stat. ( &__list. ) ;
%mend  _stat_function ;

%macro _stat_function2 ( _stat = , _start = , _end = , _var1 = , _var2 = );
  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min etc.                    ***
  *** _start and _end the start and end row to evaluate    ***
  *** _var1/1 should be the name associated to the         ***
  *** first / last column                                  **/

  %local __list __var1  __var2 ;
  %let   __var1 = %sysfunc(upcase(&_var1));
  %let   __var2 = %sysfunc(upcase(&_var2));
  %do i = &_start. %to &_end. 1;
      %do j = %sysfunc(input( "&__var1." , VARN2N. ))
          %to %sysfunc(input( "&__var2." , VARN2N. ))
          %let __list = &__list. _n [ &i.  , &j. ] , ;
      %end;
  %end;
  &_stat. ( &__list.  . ) ;
%mend  _stat_function2 ;

%macro _stat_function_list ( _stat = , _start = , _end = , _varlist = examdt )
;
  /** The parameter _stat contains a statistical function ***
  *** like N, Nmiss, sum, max, min etc.                    ***
  *** _start and _end the start and end row to evaluate    ***
  *** _varlist should be the names associated to the       ***
  *** columns wanted                                       **/

  %local __list __num __i __var ;
  %let   __var = __var;
  %let   __vlist = %sysfunc(upcase(&_varlist.));
  %let   __vlist = %sysfunc(strip(&__vlist.));
  %let   __num   = %sysfunc(countw(&__vlist.));
  %put Selected list of variables : &__vlist. ;
  %put &__num. columns selected for funktion &_stat.;
```

```
  %do __I = 1 %to &__num.;
      %let __var = %qscan( &__vlist. , &__i. );
      %put The &__i.th selected variable is &__var. ;
      %do __j = &_start. %to &_end.;
         %let __list = &__list. _n [ &__j.  , input( "&__var." , VARN2N. ) ] ,
;
      %end;
  %end;

  &_stat. ( &__list.  . ) ;
%mend  _stat_function_list ;


/* Compare the number of observations of teh result with the example before
 I If you want to play just change the _by option in the second macro call to
BP_STATUS
%global all_char all_num;

%def_loop( _dsin=  _tst
         , _dsdef= _tst2
         , _by=  SMOKING_STATUS WEIGHT_STATUS BP_STATUS AGEATSTART
         , _nrows= &r.
         , _ncols= &n_cols.
         , _ccols= &c_cols.
         , _c_new= c_test
         , _n_new= n_test
         , _c_cnt= all_char
         , _n_cnt= all_num
         , _clength=&char_len. );

    n_test = _row;
    c_test = put(_row , best. ) !! ' test';

  ** Try out the macros above **;


%end_loop( c_all_cnt= &all_char.
         , n_all_cnt= &all_num.
         , _by = AGEATSTART
         , _rc = rc );
  /*  */.
```

CONTACT INFORMATION

Raimund Storb
Boehringer Ingelheim Pharma KG
Binger Str. 173, 55216 Ingelheim am Rhein
Work Phone: +49(6132)77-93409
Email: raimund.storb@boehringer-ingelheim.com