Paper 042-31

%IFN - A Macro Function

Chang Y. Chung, Princeton University, Princeton, NJ
Ian Whitlock, Kennett Square, PA

ABSTRACT

We implement SAS® Release 9 **DATA** step **IFN()** function as a **DATA** step function-style macro, **%IFN()**, so that both the release 8 and 9 users can obtain the benefit of utilizing succinct conditional expressions in their code. In doing so, we present a general technique of using **PUT()** and **INPUT()** functions in order to detect and properly handle missing values in the input expression without generating unwanted messages in log.

"FUNCTION-STYLE" MACROS

When a macro generates a **DATA** step expression, we classify the macro as a "function-style" macro because at SAS execution time, the resolved expression is evaluated into a **DATA** step value. Such macro functions are often used to hide details of a complicated expression. A classic example is the well-known age-calculation macro function by William Kreuter[1]:

```
%macro age(begdate,enddate);
  (floor((intck('month',&begdate,&enddate)-(day(&enddate)<day(&begdate)))/12))
%mend age;
```

You can use it in this way:

```
data _null_;
  birthday = '13mar1966'd;
  today = today();
  ageToday = %age(birthday, today); /* a macro function "call" */
  format birthday today date.;
  put (_all_) (=);
run;
/* on log
birthday=13MAR66 today=05AUG05 ageToday=39
*/
```

A macro function can also be called in any step where an expression is expected. For example:

```
proc sql;
  select birthday         format=date.
       , today() as today format=date.
       , %age(birthday, today) as ageToday
  from   one
  ;
quit;
/* on lst
birthday    today  ageToday
-------------------------------
 13MAR66  05AUG05        39
*/
```

Writing this form of "function-style" macro is tricky because no SAS statements, e.g. **IF/ELSE**, can be generated by the macro. For example, in **%AGE**, the subtraction term, **(day(&enddate)<day(&begdate))**, is 0 or 1 depending on the truth or falsity of the logical expression, i.e. there is an implied branching in the formula. It

[1] See Muhlbaier(1995) for the history of this macro. See also Kreuter(1998), Whitlock(1999), and Whitlock(2000) for the detailed exposition. Whitlock(2000) also discusses "using SAS macros to extend the SAS language of functions."

was William Kreuter's insight that the correct subtraction is only critical in the birth month. To see his insight, consider the twelfth month. If the birthdate has not yet been reached, then the age is not yet one and subtracting one month gives eleven months, i.e. the age is not yet one. Now consider, one month before or after. In the former case, subtracting one doesn't matter because it cannot change the age to one. In the latter case, subtracting one does not matter because twelve months is still be one. This means that the formula not need to contain the month; hence no **IF/ELSE** statements introducing the month are needed. However, it was fortunate that the branching took such a simple form that it could be expressed in the form of subtracting either 0 or 1 in the correct place.

SAS release 9 introduced two new functions, **IFN()** and **IFC()**[2], which allows functional branching, thus one can obtain the branching ability in a macro function to provide two separate unrelated expressions. Moreover the use of **IFN()** makes it clearer to the reader that this is what is being done, than burying the branch information in logical expression used as an arithmetic one.

IFN()

The **IFN()** function returns different values based on the logical value of the conditional expression. It accepts an optional expression to be evaluated when the conditional expression is evaluated to a missing value. The full syntax is:

```
IFN (logical-expression, value-returned-when-true, value-returned-when-false<, value-returned-when-missing>)
```

The advantage of this function is that they can be used in places where **IF/ELSE** statements cannot, e.g. in **PROC SQL**. For example:

```
proc sql;
     select name
          , sex
          , ifn(sex="F", 1, 0) as female
     from sashelp.class;
quit;
/* on lst
Name      Sex    female
-----------------------
Alfred    M           0
Alice     F           1
Barbara   F           1
Carol     F           1
Henry     M           0
James     M           0
...
*/
```

We have a word of caution about logical expressions. Suppose **X** has the value 0, then the logical expressions:

```
3 / x
```

and

```
x + .
```

are missing while their truth value is 0. The **IFN()** function gives precedence to the missing value and chooses the corresponding missing-expression when it is supplied. However, in the first case the **_ERROR_** is set to one because of the division by 0. In the second case **_ERROR_** is not set, but the message about generating missing values is issued.

---

[2] Both the functions return different values based on the logical value of the conditional expression. The **IFN()** returns a numeric value, while **IFC()** returns a character value.

What if you are stuck with Release 8, then? Our **%IFN()** macro automatically utilizes the **IFN()** function if it is available, otherwise, it uses character manipulation to closely replicate what the function does.

HOW TO USE %IFN()

It is very easy to use **%IFN()** macro once you compile the macro source (either with a **%INCLUDE** statement to the macro, directly adding it to the program, or putting it in the **AUTOCALL** library). Then you just use it as if it were the **IFN()** function[3]. The first three parameters are required. The fourth is optional. Here is the PROC SQL example written using **%IFN()** macro (assuming that the source of the macro is in the file, IFN.SAS.

```
%include "ifn.sas";
proc sql;
  select name
       , sex
       , %ifn((sex="F"), 1, 0) as female
  from sashelp.class;
quit;
```

It is highly recommended to put a pair of parentheses around your parameters, especially the logical-expression part. In the above example, if there were no parentheses surrounding the logical expression, then the macro processor would understand **SEX** as a parameter because of the equal sign and complain that **SEX** had not been defined.

The same cautions about handling logical expressions containing missing values for the **IFN()** function are applicable to the **%IFN()** macro.

HOW IT WORKS

The implementation of the macro is straightforward. Here is the source code.

```
%macro ifn ( logical , true , false , missing ) ;
%*------------------------------------------------------------------------
   replicates the release 9 ifn() function.
   if called in release 9, it uses ifn() function. otherwise, it
   resorts to numeric-char-numeric trick to mimic what ifn() does

   note 1:
     it is strongly recommended to put the arguments within a
     pair of parentheses, e.g. female = %ifn( (sex="F"), 1, 0)
   note 2:
     as with ifn() function, when there is no fourth parameter,
     specified, then it returns the false choicee when logical
     is evaluated to a missing value
   by chang y. chung and ian whitlock on aug. 2005
------------------------------------------------------------------------* ;

  %local list choice retvalue msg ;

%*------------------------------------------------------------------------* ;
%*-- first three parameters are required                                --* ;

  %let retvalue = . ;

  %if ( %superq ( logical ) = %str() ) %then
    %let msg = Logical-expression ;

  %if ( %superq ( true ) = %str() ) %then
```

[3] There is one minor difference in version 8. When the logical expression contains missing values, the message about generating missing values is not issued as it would be in with the **IFN()** function in version 9.

```
  %do ;
    %if ( %superq ( msg ) ^= %str() ) %then
        %let msg = &msg, ;
    %let msg = &msg Value-returned-when-true ;
  %end ;

  %if ( %superq ( false ) = %str() ) %then
  %do ;
    %if ( %superq ( msg ) ^= %str() ) %then
        %let msg = &msg, ;
    %let msg = &msg Value-returned-when-false ;
  %end ;

  %if ( %superq ( msg ) ^= %str() ) %then %goto retrn ;

  %if ( %superq ( missing ) = %str() ) %then
    %let missing = &false ;

%*-------------------------------------------------------------------* ;
%*-- if release 9 or later, then just call the ifn() function      --* ;
  %if %sysevalf( &sysver >= 9.0 ) %then
    %let retvalue =
      ifn ( ( &logical ) , ( &true ) , ( &false ) , ( &missing ) )
    ;

%*-------------------------------------------------------------------* ;
%*-- if release 8 or ealier, then use the numeric-char-numeric     --* ;
%*-- trick to mimic what ifn() does                                --* ;
  %else
  %do ;

    %let list = put ( ( &true    ) , best12. )
             || put ( ( &false   ) , best12. )
             || put ( ( &missing ) , best12. ) ;
    %let choice = 1 + 12 * ( ( &logical ) = 0 )
                    + 12 * missing ( &logical ) ;
    %let retvalue = input ( substr ( &list , &choice , 12 ) , best12. ) ;

  %end ;

%*-------------------------------------------------------------------* ;
%*-- if error, then return a missing, set _ERROR_ to 1             --* ;
%retrn:
  %if ( %superq ( msg ) ^= %str() ) %then
    %put ERROR: (ifn) Missing required parameter(s): &msg.. ;

  &retvalue

%mend  ifn ;
```

The macro code has four sections. The first section checks if the three required parameters are present. If not, then the control is immediately moved to the fourth and the last section, which generates an error message and returns a missing value. The second section is for SAS release 9. In this case, the macro just hands the job out to the **IFN()** function.

The third section implements the core programming technique, a double type conversion. The first conversion is from numeric to character by the **PUT()** function. It forces character string values to have a common length in a concatenated list of the values. Then the **SUBSTR()** function is used to extract the chosen value. The second conversion converts the chosen value back to numeric by the **INPUT()** function. To see this operation in detail suppose that the macro call was:

```
%ifn((sex="F"), 1, 0)
```

The above (omitting the missing complication) resolves to the expression:

```
input ( substr ( put ( ( 1 ) , best12. ) || put ( ( 0 ) , best12. )
               , 1 + 12 * ( (sex="F") = 0 )
               , 12 )
      , best12. )
```

The true and false values are converted into a string by two PUT( ) statements and concatenated:

```
input ( substr (“1          2          ”
               , 1 + 12 * ( (sex="F") = 0 )
               , 12 )
      , best12. )
```

Suppose that the variable SEX has a value of “F”. Then the logical expression, SEX=”F” is evaluated True or 1. Thus, the part goes through the logical evaluation as so:

```
( (sex="F") ) = 0 )
```

evaluates to (0) because **SEX="F"** is 1.

So, the expression as a whole now looks like:

```
input ( substr (“1          2          ”
               , 1 + 12 * ( 0 )
               , 12 )
      , best12. )
```

This reduces to:,

```
input ( substr (“1          2          ”, 1, 12 ), best12. )
```

or,

```
input ( “1          ”, best12. )
```

or

```
1
```

To handle the case where the logical expression is missing remember that logical expressions are just numbers where 0 and missing. We can distinguish the 0 case from the missing case by adding

```
12 * ( missing ( logical-expression ) )
```

to the second parameter of the **SUBSTR( )** function.

SUMMARY

After briefly reviewing the concept of a macro function, we presented a new macro, %IFN( ), which faithfully implements the functionality of the SAS release 9 function IFN( ). The macro resolves to a **DATA** step expression which can be used in any step where such expressions are acceptable. The macro features a useful programming technique, the double type conversion, to make a choice between three arithmetic expressions based on whether a logical expression is false, true, or missing.

REFERENCES

Gang, Hua (2005). “quick data step.” A SAS-L posting on Jul 19, 2005. Archived at <http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0507C&L=sas-l&D=1&H=0&O=A&T=1&P=33473>.

Kreuter, William (1998). “Sample 271: Accurately Calculating Age with Only One Line of Code.” A SAS Sample posted on support.sas.com. <http://support.sas.com/ctx/samples/index.jsp?sid=271>.

Kreuter, William (1993). “Re: Computing Age in years.” A SAS-L posting. Archived at Marist. <http://vm.marist.edu/htbin/wlvtype?SAS-L.24262>.

Muhlbaier, Lawrence H. (1996). “Re: strictly correct age calculation?” A SAS-L posting on May 7, 1996. Archived at <http://www.listserv.uga.edu/cgi-bin/wa?A2=ind9605A&L=sas-l&P=R9691&D=1&H=0&O=D&T=1>.

Whitlock, Ian (2000). "SAS Talk." Column in DCSUG News Letter. September 2000. <http://www.dc-sug.org/dcsep00.pdf>.

Whitlock, H Ian (1999), "Managing the INTNX Function," in the Proceedings of the 1999 SouthEast SAS Users Group Conference.

Whitlock, Ian (2005). "Re: quick data step." A SAS-L posting on Jul 20, 2005. Archived at <http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0507C&L=sas-l&D=1&H=0&O=A&T=1&P=37239>.

ACKNOWLEDGEMENTS

We would like to thank Hua Gang for posting an interesting question on SAS-L (Gang 2005). In responding, Ian (2005) generalized Gang's problem and made it a programming challenge, out of which grew this paper.

CONTACT INFORMATION

Chang Y. Chung
Princeton University
#216 Wallace Hall
Princeton, NJ 08540
cchung@princeton.edu
http://changchung.com/

Ian Whitlock
29 Lonsdale Lane
Kennett Square, PA 19348
ian.whitlock@comcast.net