Getting the IN operator to FUNCTION inside a SAS® Macro

Perry Watts, Independent Consultant, Elkins Park, PA

ABSTRACT

Large SAS macros with many parameters require that user values be checked at run time. One convenient checking tool is the IN operator that has been extended in 9.2 SAS to work inside a macro. Usage and syntax updates for the newly extended operator are included in the presentation.

For those with earlier versions of SAS software, the INFCN ("in-function") macro has been developed to mimic the 9.2 IN operator. INFCN has been selected as a name, because the INDEXW function plays a major role in its operation. Also the macro has been implemented as a macro function so that it ends up working just like an operator. The crossover from function to operator is made possible by the fact that these structures can often be used interchangeably. In the paper, the interchangeability is demonstrated and the INFCN macro is reviewed in detail.

PROBLEM DEFINITION

In Building the Better Macro: Best Practices for the Design of Reliable, Effective Tools Frank DiIorio argues for consistency when using keyword parameters in a system of interrelated macros:

Just as the presence of keyword parameters makes the macro more usable, so does consistency of how they are named and given values within a system of macros. Parameters that identify similar actions or content should be named consistently. Parameter values should also be coded consistent. [3, p.4] (italics added).

From the quote, consistency among a set of macros not only applies to parameter names but also to their values. For example, DiIorio argues for a universal "yes" when assignments are made to boolean parameters: not a "yes" for one and a "t" for another.

While there is no doubt that consistency improves the design and reliability of reusable software, few developers have the authority to restrict values supplied to macro parameters. While users have no choice but to adhere to default definitions, they typically want their macros to be "flexible". For example, in a graphics macro developed a few years ago, a percent bar chart had to be produced when any of the following were supplied to the CHARTTYPE parameter:

PCT, CLASSPCT, GROUPPCT, SUBGROUPPCT, OAPCT, OTHERPCT, PERCENT, __PERCENTAGE, PERCENTAGE, RPCT, MP100PCT, G100PCT, or SG100PCT

If the IN operator worked in version 8 SAS Macro Language, this specification would have been easy to implement. However, here is the error message that is returned when version 9.2 SAS is not being used:

ERROR: A character operand was found in the %EVAL function or %IF condition where a numeric operand is required. The condition was:
RPCT in PCT, CLASSPCT, GROUPPCT, SUBGROUPPCT, OAPCT, OTHERPCT, PERCENT, __PERCENTAGE, PERCENTAGE, RPCT, MP100PCT, G100PCT, SG100PCT

An application of %SYSFUNC doesn't work either. IN is an operator, not a function. Also, reliance upon %IF-%THEN-%ELSE statements is beyond the pale - especially in a macro that supports many such "flexible" parameters. Formats can provide a workaround, but they are unwieldy too.

THE IN(#) OPERATOR WORKS IN A SAS 9.2 MACRO

System options need to be defined to get the IN (or #) operator to work inside a SAS 9.2 macro. MINOPERATOR (for Macro-In-Operator) must be turned on, because the default is NOMINOPERATOR, and if the delimiter in the match list is not a space, it too must be specified with MINDELIMITER= [6,p.322-3]. The following, therefore, should work:

```sas
options MINOPERATOR;
%let excerpt=PERCENT;
%let matchList=PCT CLASSPCT GROUPPCT SUBGROUPPCT OAPCT OTHERPCT PERCENT, __PERCENTAGE, PERCENTAGE, RPCT, MP100PCT, G100PCT, or SG100PCT
%put %eval(&excerpt in &matchList) /* Output = 1 */
```

DIFFERENTIATING BETWEEN COMPARISON OPERATORS AND FUNCTIONS IN SAS

Comparison operators in SAS include EQ, NE, GT, GE, LT, LE as well as IN [4,p.118-122]. Operators are used to evaluate the relationship between two operands, returning a ‘1’ if the expression is TRUE, and a ‘0’ if the expression is FALSE.
Functions differ in structure from operators. In SAS, a function operates on supplied arguments to perform a calculation or operation that is returned as a value \[4,p.39\]. Arguments always follow the function name and are enclosed in parentheses. A listing of SAS-supplied functions can be found in any version of the SAS Language Reference Dictionary manuals whereas user-defined functions are only available in the SAS macro language. For functions, the number of arguments can vary whereas comparison operators are nested between two operands.

Despite their differences, operators and functions both provide return values. If the return value from a function is any non-zero number, SAS will treat it as TRUE when the function is used in an IF statement. Only zeroes are defined as FALSE. While the return values of '0' and '1' are more subtle for operators, they can, nevertheless, be inferred in the source code that follows:

```sas
%macro TestIF;
%let A=3; %let B=2;
%if &A GT &B %then %do;
  %put Implicit: A is greater than B;
%end;
%if &A GT &B EQ 1 %then %do;
  %put Explicit: A is greater than B;
%end;
%if &B GT &A %then %do;
  %put Implicit: B NOT greater than A IS NOT PRINTED;
%end;
%if &B GT &A EQ 0 %then %do;
  %put Explicit: B is NOT greater than A;
%end;
%mend TestIf;
```

Output to the LOG from the macro:

Implicit: A is greater than B
Explicit: A is greater than B
Explicit: B is NOT greater than A

Functions do not belong exclusively to the function. In Base SAS, the IN also operator uses them. This means that the IN operator and the INDEXW function have a similar appearance, and they can be used interchangeably to generate the same output from the data step below:

```sas
data test;
  length A $10;
  A='xyzzy';
  put 'By Operator';
  if A in('deffde','xyzzy') then
    put 'IMPLICIT: A = deffde OR A = xyzzy';
  if A in('deffde','xyzzy') NE 0 then
    put 'EXPLICIT: A = deffde OR A = xyzzy';
  if A in('deffde','abcdef') eq 0 then
    put 'EXPLICIT: A ^= deffde and A ^= abcdef ';
  put 'BY Function';
  if indexW('deffde xyzzy', A) then
    put 'IMPLICIT: A = deffde OR A = xyzzy';
  if indexW('deffde xyzzy', A) NE 0 then
    put 'EXPLICIT: A = deffde OR A = xyzzy';
  if indexW('deffde abcdef', A) eq 0 then
    put 'EXPLICIT: A ^= deffde and A ^= abcdef ';
run;
```

Output to the LOG from data set TEST is identical for both the IN operator and the INDEXW function:

By Operator
IMPLICIT: A = deffde OR A = xyzzy
EXPLICIT: A = deffde OR A = xyzzy
EXPLICIT: A ^= deffde and A ^= abcdef

BY Function
IMPLICIT: A = deffde OR A = xyzzy
EXPLICIT: A = deffde OR A = xyzzy
EXPLICIT: A ^= deffde and A ^= abcdef

Parentheses do not belong exclusively to the function. In Base SAS, the IN also operator uses them. This means that the IN operator and the INDEXW function have a similar appearance, and they can be used interchangeably to generate the same output from the data step below:

```sas
%macro TestIF;
%let A=3; %let B=2;
%if &A GT &B %then %do;
  %put Implicit: A is greater than B;
%end;
%if &A GT &B EQ 1 %then %do;
  %put Explicit: A is greater than B;
%end;
%if &B GT &A %then %do;
  %put Implicit: B NOT greater than A IS NOT PRINTED;
%end;
%if &B GT &A EQ 0 %then %do;
  %put Explicit: B is NOT greater than A;
%end;
%mend TestIf;
```

Output to the LOG from the macro:

Implicit: A is greater than B
Explicit: A is greater than B
Explicit: B is NOT greater than A

---

* if indexW('deffde xyzzy', A) translates to if 8 or TRUE. In this example, a function call is being used as a conditional expression, since logical values in SAS are numeric.
Although functions and operators have a lot in common, assignment statements can only be used to store return values from functions, not operators. For example, the return value from executing indexW above was retrieved with:

\[
x = \text{indexW}('deffde xyzzy', A); /* x = 8 */
\]

As a workaround, SAS nests operators within function calls to retrieve their output! In the data step below, a value of 1 is assigned to XX with an application of IFN that mimics the IF/THEN/ELSE construct [5,p.586]:

```sas
data IfTrue;
  A = 3; B = 2;
  XX = ifN(A GT B, 1, 0);
run;
```

For the SAS macro language, the corresponding function is %EVAL:

```sas
%let XX = %eval(&A GT &B);
```

THE INFCN MACRO FUNCTION

Given that return values from functions can be used in logical expressions makes it possible to solve the IN problem with INDEXW inside a macro. The INFCN macro is listed in full below:

```sas
%macro InFcn (excerpt, matchlist);
  %local TorF quotes;
  %let quotes=%str('%');
  %let excerpt = %nrbquote(%sysfunc(compress(&excerpt,"&quotes")));
  %let matchlist = %sysfunc(translate(&matchList,' ',','));
  %let matchlist = %nrbquote(%sysfunc(compress(&matchList,"&quotes")));
  %let TorF = %SYSFUNC(indexW(%upcase(&matchlist), %upcase(&excerpt)));
  &TorF
%mend InFcn;
```

1. **InFn** is a macro function. Art Carpenter lists the following characteristics of macro functions in his SUGI 27 paper *Macro Functions: How to Make Them - How to Use Them* [2]:
   - All statements in the macro must be macro statements beginning with a percent sign (%) or ampersand (&).
   - The macro should create NO macro variables other than those that are local to that macro. (Macro parameters are local by definition).
   - The macro should resolve to the value that is to be returned.

The two positional parameters associated with INFCN are defined in IN-operator order; i.e. EXCERPT IN MATCHLIST translates to %INFcn(EXCERPT,MATCHLIST). For the INDEXW function, the order is reversed: indexW(MATCHLIST,EXCERPT).

2. **Local macro variables are declared in the macro function.**
3. **Single instances of single and double parentheses are masked with a % sign such that quotes = '\"'.**
4. **Working from the inside out:**
   - Quotes are not used to define character strings in the macro language. Therefore they are removed with the SAS COMPRESS function that is made available through SYSFUNC.
   - NRQUOTE (no-resolve blind quote) [1, p.155-159] masks character strings (NE) and macro symbols (%)
5. **In Base SAS, the delimiter for the match string in the IN operator is the comma whereas it is a space in the INDEXW function. Therefore, commas are replaced with spaces with an application of TRANSLATE.**
6. **Again, working from inside out:**
   - Text is made case insensitive with the application of %UPCASE on both EXCERPT and MATCHLIST.
   - The IndexW function returns a number between 0 and the position of the first character of the last word in the match-list. If the number equals zero, the expression will evaluate to FALSE, otherwise any positive integer will be treated as TRUE. The return value is assigned to the macro variable, TORF.
7. **Here is where TORF is actually returned - with an ampersand. Note the absence of a semicolon. The presence of one would cause problems in the calling program.**

INVOKING INFCN

In the first example, the problem with the assignment to CHARTTYPE described in the opening paragraphs is solved:

```sas
%macro getBarChartType(thisType=, allTypes=);
  %local position chrtType;
  %let position = %InFcn(&thisType,&allTypes);
  %put position = &position;
  %if %InFcn(&thisType,&allTypes) %then %let chrtType=PCT;
  %else %let chrtType=FREQ;
  &chrtType
%mend getbarchartType;
```
%let chrtType=%getbarchartType(
    thisType=, 
    allTypes= % PCT CLASSPCT GROUFPCT SUBGROUFPCT OAPCT OTHERPCT PERCENT 
        _PERCENTAGE PERCENTAGE RPCT MP100PCT G100PCT SG100PCT );

%let chrtType=%getbarchartType(
    thisType="classPct",
    allTypes= %nrStr('%','pct','classpct','grouppct','subgrouppct',
        'oapct','otherpct','percent','__percentage',
        'percentage','rpct','mp100pct','g100pct','sg100pct' ) );

%let chrtType=%getbarchartType(
    thisType=count,
    allTypes= % PCT CLASSPCT GROUFPCT SUBGROUFPCT OAPCT OTHERPCT PERCENT 
        _PERCENTAGE PERCENTAGE RPCT MP100PCT G100PCT SG100PCT );

• %macro getBarChartType

The calling macro, GETBARCHARTTYPE, is also a macro function. In GETBARCHARTTYPE, INFCN is used first in an assignment statement and then in a conditional expression. In run #1, the % sign is successfully managed. With POSITION=1 (TRUE), PCT is assigned to CHARTTYPE. In run #2, mixed text parameters, commas, and quotation marks are also handled so that POSITION=7 (TRUE) and PCT is again assigned to CHARTTYPE. In run #3, COUNT cannot be found in the match string. Therefore, POSITION=0 (FALSE), and FREQ, the default value in this example, is assigned to CHARTTYPE.

In the second example, the IN operator and the INFCN macro function are compared in two data steps that generate the same output.

```
data one; /* WITH IN OPERATOR */
    length ingroup $1;
    do i=1 to 10;
        if i in(1,4,5) then ingroup='Y';
        else ingroup='N';
    output;
    end;
run;
title1 'Data: One';
proc print data=one;
run;
```

```
%macro enumerate;
    data OneAlso;
        length ingroup $1;
        %do i= 1 %to 10;
            i=&i;
        %if %infcn(&i,1 4 5) %then %str(ingroup='Y';);
            %else %str(ingroup='N';);
        output;
        %end;
run;
title1 'Data: OneAlso';
proc print data=OneAlso;
run;
%mend enumerate;
```

The conclusion for this brief paper is an exercise: Review the source code above and explain why data sets ONE and ONEALSO are the same. Use concepts drawn from the paper.

REFERENCES


SAS INSTITUTE REFERENCES:

SAS-L DISCUSSIONS ABOUT THE USE OF THE IN OPERATOR FOR MACROS:

SAS-L@LISTSERV.UGA.EDU. Re: Help with Macro. Posted by Toby Dunn on 11 May 2007. Replaces the %INDEX macro function with the INDEXW SAS function for evaluating the match string in his macro, FINDIN. FINDIN uses %EVAL to return a value to the calling program - working, in effect, just like a macro function.

SAS-L@LISTSERV.UGA.EDU. Re: Macro to convert Dates to 8601 format. Posted by Jim Groeneveld on 4 Nov 2008. He provides a link to his IN macro that also is a macro function. Groeneveld's IN macro uses a do-loop instead of INDEXW to check excerpt against match string.

WHAT'S IN THE NESUG 2009 PROCEEDINGS OR AVAILABLE BY REQUEST:
1) The INFCN macro function (inFcn.sas)
2) The calling program used for invoking INFCN (CallinFcn.sas)

TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

The author welcomes feedback via email at perryWatts@comcast.net.