Paper 022-2009

IS THIS MACRO PARAMETER BLANK?

Chang Y. Chung, Princeton University, Princeton, NJ
John King, Ouachita Clinical Data Services, Mount Ida, AR

ABSTRACT

The first thing most macros do is checking input parameters. Still, many macro users are unsure when it comes to implementing the parameter checks. This is the case even with the most elementary form, namely the check of the existence of a parameter value, or the "Is-Blank" check. We review and compare various ways to implement it. We then recommend one that performs well given various input parameter values, packaged in a function-style macro %ISBLANK(), which returns 1 when the parameter value is "blank," and 0 otherwise.

INTRODUCTION

Suppose that we are writing a macro that does something given a parameter. Implementing the macro, we should check if the parameter is "blank" or not. How do you do this? A simple way may be as below:

```
%macro test(param);
  %if &param eq %then %do;
    %put error: param is blank.;
    %return;
  %end;
  %put note: param is OK.;
%mend  test;
```

The test works. For example, it works for these cases:

```
%test()
/* on log
error: param is blank.
*/
%test(abc)
/* on log
note: param is OK.
*/
```

But it does not work all the time. An innocent-looking parameter value can cause an unexpected error:

```
%test(or)
/* on log
ERROR: A character operand was found in the %EVAL function or %IF condition
where a numeric operand is required. The condition was: &param eq
ERROR: The macro TEST will stop executing.
*/
```

There must be a better way. We reviewed many macros and, to our surprise, there does not seem to be a standard. As with Scott (2008), we also find many different implementations. Even after discarding the obviously incorrect ones, we end up with eight. In this paper, we introduce, compare, and test them, in the hope that we may recommend one that is suitable under general circumstances.

ASSUMPTIONS

Different macros check parameters differently because they have different expectations about the input values. Macro documentation is often not explicit enough on the assumptions about the input parameter and its value. Here are a couple of *our* assumptions about the input parameter:

(1) Responsibility of macro author starts only after a *successful* macro invocation. In other words, a macro writer cannot be responsible when the macro invocation itself fails. This can happen in many ways: Users may supply an unexpected number of parameters; Macro invocation may resolve to un-quoted special characters; Macro or parameter names are spelled incorrectly. Here are some of the examples of such failures:

```
%tast(,)                                        /* 1 */

%test(chang's favorite)                         /* 2 */

%let var = (;
%test(&var)                                     /* 3 */
```

In the 1 above, the macro name is misspelled; SAS will not be able to find the macro. Or worse, it will invoke a "wrong" one, if it happens that such a macro exists. It also passes two "blank" parameters to the macro, while the macro may be expecting only one. In 2, the apostrophe signals the beginning of a single-quoted string which is not terminated; This will wreak a major havoc on the current session. The error in 3 is more subtle but is as deadly as 2; The macro invocation looks all right at the first glance but the macro variable VAR is resolved to an unmatched parenthesis.

This kind of failures is difficult to prevent and the macro author has little means to prevent them. The macro processor, in fact, constructs the complete macro invocation even before the macro starts to execute.

(2) The parameter value may or may not be macro quoted before it is passed into the macro. This complicates a bit what we mean by the parameter being "blank." Here we assume the following:

a. Nothing(or "null") is a blank

b. One or more space characters (BYTE(32) in ASCII collating sequence) are blank.

We also assume that:

c. An input parameter value contains only the "printable" characters (BYTE(32) through BYTE(255) in ASCII collating sequence) and optionally the macro quotation delta characters

These assumptions imply that all the macro calls below result in the PARAM being blank:

```
%test()                                     /* nothing */

%test(%str( ))              /* a macro quoted space char */

%test(param=)                           /* nothing again */

%let var=;
%test(%superq(var))      /* nothing quoted by %SUPERQ() */
```

On the other hand, the assumption c above makes our macro's behavior in the following situation *un-defined*, because it involves unprintable characters in the parameter values:

```
data _null_;
  length value $1;
  name = "LF";
  value = byte(10); /* assuming an ascii machine */
```

```
  call symput(name, value);
run;
%test(&LF)
```

CONTENDERS

After reviewing many existing macros, we choose following eight implementations of the Is-the-Parameter-Blank check. In no particular order, they are:

```
%if &param eq %then ...                          /* C1 */

%if %bquote(&param)= %then ...                   /* C2 */
%if %nrbquote(&param)= %then ...                 /* C3 */
%if %superq(param)= %then ...                    /* C4 */

%if "&param" = "" %then ...                      /* C5 */

%if %length(&param) = 0 %then ...                /* C6 */
%if %length(%qleft(%qtrim(&param))) = 0 %then .../* C7 */

%if %sysevalf(%superq(param)=,boolean) %then ... /* C8 */
```

C1 is the basic testing with no macro quoting at all. C2 through C4 use a different macro quoting function in turn. We omit %QUOTE() and %NRQUOTE() functions because their usage is discouraged except for the unique quoting needs and for the backward compatibility reasons.[1] C5 double-quotes the parameter value. C6 and C7 are based on the %LENGTH function. C7 is quite elaborate and found in the %DATATYP macro included in the SAS autocall library. C8 is similar to C4 but evaluates the logical expression explicitly within the %SYSEVALF() function. There are other variations of the above, and we will come back to some of them when we discuss the test results.

(CON)TEST SETUP

Our test is simple. We write a test macro with above "contenders." Call the macros with various parameter values. Then see what happens. We pay close attention to: If the test macro errors or not; If the test macro takes exceptionally long time to run; If the test result is indeed correct. Some input values may hang the SAS session or cause harmful side effects. In order to prevent unwanted interaction, we run each test in its own separate SAS session.

The input parameters are chosen carefully and are to reflect various extreme cases. They include: nothing, quoted blanks, letters, numbers, special characters, logical operators, very long character strings, and a very long string of quoted blanks, and a large number. The actual test codes are automatically generated, submitted, and the log analyzed. The source code is presented in the appendix. All the tests are run using SAS® for Windows Release 9.1.3 with SP4 (TS1M3) on Microsoft® Windows® Vista™ Ultimate (Version 6.0, Service Pack 1).

THE WINNER

Table 1 shows the test results. An empty cell represents that the test macro performs as expected. Certain input values cause C1 and C2 to error out. C5 and C6 sometimes return incorrect results. C7 sometimes takes

[1] See the *Note* at the bottom of page 77 of the SAS Institute Inc. (2008) *SAS® 9.2 Macro Language: Reference.*

exceptionally long time to run. This leaves, C3, C4 and C8. But a large number causes overflow error on %EVAL(), and this knocks out C3 and C4, leaving C8 a winner.

We make our recommendation accessible by providing a simple macro, %ISBLANK():

```
%macro isBlank(param);
  %sysevalf(%superq(param)=,boolean)
%mend  isBlank;
```

Table 1. Test Results

Contender / *Input value*	*C1*	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*	*C8*
1. (nothing)								
2. %str( )					Incorrect	Incorrect		
3. &	Error	Error						
4. A>B+C>D	Error							
5. %bquote(Tom&Jerry)	Error	Error					Warning	
6. %nrstr(%()								
7. 1 or	Incorrect							
8. (a long string – 65533 characters)							Error	
9. (10,000 quoted space characters)					Incorrect	Incorrect	Takes long time	
10. (a large number) 43876106244387610624	Error	Error	Error	Error				

DISCUSSION

We discuss several topics related to the results shown in the Table 1. We start with a brief discussion on how the macro quoting works. After that, we review each contender one by one. Then, we discuss briefly four related issues in turn. They are: Logical Operators versus Mnemonics, Macro Reference First versus Logical Operator First, Maximum Length of Macro Text Expression, and finally Other Variations.

Not a small part of the discussion is on how the %EVAL function works, since it is the implicit %EVAL function that evaluates the expression between %IF and %THEN. For a more comprehensive exposition on %EVAL, see the excellent paper by Xu and Zhang (2004).

Macro Quoting and Delta Characters

Macro quoting is needed because the macro processor must know whether to interpret a particular special character as text or as a symbol in the macro language. Suppose that the processor encountered a line, "%change." It needs to know if it is a call to a macro named change, or just a text string, possibly a shortcut of, "percent change."

Without quoting, the macro processor assumes the first and tries to call the macro. This may or may not be what you want. Macro quoting makes it possible to process "%change" as just a text string. You can also quote when a macro is compiled or while the macro is executing.

We quote with a macro quoting function. There are different quoting functions. Roughly speaking, they differ by: (1) when they quote; and (2) what they quote.

When you call a macro quoting function, it returns a text string with its special characters and mnemonics *masked*. The text is also sandwiched by a pair of *delta characters* (leading and trailing blanks are preserved). The one in front (prefix delta character) signals the beginning of the macro quoted string. It also let the macro processor know what kind of macro quoting is applied. The delta characters are never alphabets or digits, but a byte character not found on the keyboard. They are not portable from a platform to another.

As we saw earlier, macro quoting is not an option in implementing a general parameter checks. One reason why users are unsure of implementing parameter checks, then, is because there are so many different ways to quote the input parameters. One goal of this paper is to see which one does a reasonable job for our specific purpose.

For an excellent exposition of SAS macro system (including the detailed discussion of the delta characters), see *SAS 9.2 Macro Language Reference* (SAS Institute, 2008).

Contenders

C1 and C2

```
%if &param eq %then ...                          /* C1 */
%if %bquote(&param)= %then ...                   /* C2 */
```

It is not surprising to see that C1 fails, since its input is not protected. What may be surprising is that C2 has any vulnerability at all, since C2 is much used. For example, the %DS2CSV autocall macro and the %POWERRXC macro by SAS Institute(2007) use C2. The macro quoting function %BQUOTE is vulnerable to a standalone ampersand (&), since %BQUOTE does not quote it, and the %EVAL considers it as a logical operator, AND. Sometimes it happens accidentally like so:

```
%let param = Brooks & Sons, Inc.;
%put %eval(%bquote(&param)=); /* C2 errs */
```

Another special character to watch out for is a single percent sign (%), which may not cause an error but a warning:

```
%let param = 120% increase;
%put %eval(%bquote(&param)=); /* C2 generates a warning */
```

C3 and C4

```
%if %nrbquote(&param)= %then ...                    /* C3 */
%if %superq(param)= %then ...                       /* C4 */
```

Both the problems disappear when %NRBQUOTE is used as in C3:

```
%let param = me % you;
%put %eval(%nrbquote(&param)=); /* C3 returns 0 */
%let param = Brooks & Sons, Inc.;
%put %eval(%nrbquote(&param)=); /* C3 returns 0 */
```

C3 relies on %NRBQUOTE and C4 on %SUPERQ. There is an important difference in terms of the calling syntax. With the %NRBQUOTE, we pass the macro variable reference as the parameter, like %NRBQUOTE(&PARAM). On the other hand, we pass the name of the macro variable to %SUPERQ, such as %SUPERQ(PARAM).

They both quote the same special characters and macro triggers. However, since the invocation of %NRBQUOTE implies a macro reference being resolved, it is inevitable that it can cause a warning. For instance:

```
%symdel jerry;
%let param = tom&jerry;
%put %eval( %nrbquote(&param)= ); /* C3 returns 0 with a warning */
%put %eval( %superq(param)= );    /* C4 returns 0 */
```

In the above, we make sure that there is no macro variable, jerry, by deleting it with %SYMDEL. Then we store a text string “tom&jerry” in the macro variable PARAM. Notice that both the C3 and C4 return the expected result. C3, however, generates a warning that &JERRY is not resolved.

C5 and C6

```
%if "&param" = "" %then ...                         /* C5 */
%if %length(&param) = 0 %then ...                   /* C6 */
```

Both C5 and C6 return incorrect results when we feed quoted space characters as the parameter value. This is expected since, with C5, we are comparing two double quoted strings of different lengths. As long as the input parameter is not null(or nothing), the comparison will be false.

To the implicit %EVAL(), the double quotation mark is just another character. On the other hand, the SAS supervisor (especially the tokenizer) is watching how long a double quoted string gets. For a long parameter value, SAS issues a warning suspecting an “unbalanced quotation marks.” Below demonstrates that the warning shows up when the quoted string gets longer than 262 characters, including the quotation marks:

```
%let param = %sysfunc(repeat(X, 260-1));
%put %eval("&param"=""); /* C5 returns 0 */
%let param = %sysfunc(repeat(X, 261-1));
%put %eval("&param"=""); /* C5 0 and warning */
```

You can turn off this warning using the NOQUOTELENMAX system option, though.

C6 also returns incorrect results when given macro quoted blanks because it checks if the input parameter is of length 0. C5 and C6, thus, are checking if the input parameter is “nothing” or “null,” instead of checking if it is “blank.” Notice that the is-null check is much stricter. You may find it is necessary in other contexts.

C6 does not require any macro quoting because the parameter reference here is used as a parameter to a macro function %LENGTH(), first. The implicit %EVAL() sees only the return value from %LENGTH(), which is just a number. Thus the quoting is only required to make the invocation of the %LENGTH() function successful, and this hurdle is already jumped over when the parameter is first passed on to the %TEST macro successfully.

C7

```
%if %length(%qleft(%qtrim(&param))) = 0 %then .../* C7 */
```

C7 is quite elaborate. It first %QTRIM's then %QLEFT's the parameter, before it is sent to %LENGTH. The purpose is to remove both the possible leading and trailing blanks before we check the length. One may notice that the order of the function calls odd, since in a data step, we hardly use LEFT(TRIM(something)) to remove both the leading and trailing blanks. We use TRIMN(LEFT(something)), or STRIP(something), instead:

```
data _null_;
  a = "  A  ";
  s3 = "***";
  leftTrim = s3 || left(trim(a)) || s3;
  trimnLeft = s3 || trimn(Left(a)) || s3;
  strip = s3 || strip(a) || s3;
  put leftTrim= trimnLeft= strip=;
run;
/* on log
leftTrim=***A  *** trimnLeft=***A*** strip=***A***
*/
```

Notice the trailing blanks in the LEFTTRIM. This is because DATA step works with fixed-width character values. The LEFT function shifts the (non-blank) values to the left and the rest is filled with blanks. Macro, however, works with varying-length strings. A return value from the %QLEFT autocall macro function does not include the blanks on the end. Thus, when it comes to macro functions %QLEFT and %QTRIM, it does not matter which function is called first.

%QLEFT locates a non-blank character in the input and %QSUBSET's from that position to the end. Thus, the trimming of the leading blanks is accomplished in a single function call. The %QTRIM function, on the other hand, removes one blank character at a time starting from the end. This involves repeated resolutions of the input parameter and when the parameter value is long, this takes quite some time. This is why C7 takes such a long time handling long quoted blanks.

C8

```
%if %sysevalf(%superq(param)=,boolean) %then ... /* C8 */
```

C8 utilizes the %SUPERQ function as C4 does. C8, however, evaluates the equality condition within a specific call to the %SYSEVALF function. %SYSEVALF is capable of floating-point arithmetic and properly handles missing values. %EVAL is limited in handling only integer calculations. If the input parameter is unquoted, then it could make a difference:

```
%let param = 1. or;              /* notice the decimal point (dot) */
%put %eval(&param=);             /* C1 causes an error */
%put %sysevalf(&param=, boolean); /* This is not C8. This returns 1 */
```

Once quoted, we are safely back in the territory of string comparisons. Unless that is, the string is a large number with lots of digits. Quoted or not, digits will be recognized as a number by %EVAL function and it may error out, if the number is too large for %EVAL.

In short, we need both the macro quotation (provided by %SUPERQ), and the capability to handle large numbers (which is what %SYSEVALF is capable of doing). So, C8 is our winner.

Logical Operators versus Mnemonics

The major differences between the logical operators (<, <=, =, ...) and their mnemonics (LT, LE, EQ, ...) within the %EVAL function is that the latter requires blanks between it and its operands. Not macro quoted, the space between the parameter reference (for example, &PARAM) and the logical operator (i.e., = or EQ) does matter.

This sometimes results in an unexpected outcome. Here are some examples with the parameter not being quoted:

```
%let param = <;
%put %eval(&param=);    /* 1 */
%put %eval(&param  =);  /* 0 */
%put %eval(&parameq);   /* warns that parameq not resolved and errors */
%put %eval(&param  eq); /* 0 */
```

It is easy to see that the third one above errors. The second and the fourth work correctly. The first one, though, does not. The problem is that the parameter is resolved to and its value is combined with the equal sign. Together, then, they are tokenized into another logical operator, <=, surrounded by nothings as its own operands. The expression, thus, evaluates as true and %EVAL() reports it as 1, which is unexpected.

Once macro quoted, though, the space between the operand and the operator becomes less meaningful:

```
%let param = <;
%put %eval(%bquote(&param)=);    /* 0 */
%put %eval(%bquote(&param)  =);  /* 0 */
%put %eval(%bquote(&param)  eq); /* 0 */
```

This implies that, once the parameter is quoted, then there is no major difference between the logical operators and their mnemonics.

To be technically more precise, the mnemonics do not need space characters per se. Any means are fine as long as they make SAS tokenizer to separate the mnemonic as a separate token. For instance, the following two work correctly:

```
%let param = <;
%put %eval(&param.eq);          /* 0 */
%put %eval(%bquote(&param)eq); /* 0 */
```

Macro Reference First versus Logical Operator First

How about the order between the reference and the operator? All the above candidates put the parameter reference before the logical operator. What happens if we put the logical operator ahead of the parameter reference? Again, our observation is that it could cause some complications especially if the parameter reference is not macro quoted:

```
%let param = 1 or;
%put %eval(&param =);         /* 1 */
%put %eval(= &param);         /* error */
%put %eval(%bquote(&param)=); /* 0 */
%put %eval(=%bquote(&param)); /* 0 */
```

The first one above resolves to “1 or =”. With the precedence of the equal-to operator (=) being higher than the OR, the expression is the same as “1 OR (=),” and it is TRUE, or 1, which is incorrect for our purpose. In the second one above, the expression resolves to “(=1) or” and the “(=1)” part is FALSE(0), but the OR expects either 0 or 1 on the right side of itself. Thus an error. This is an interesting case, since both the first and the second fail to provide an expected result. Once macro quoted, though, we get what we want, and the question of reference first versus operator first loses technical relevance.

Maximum Length of Macro Text Expression

As of SAS Release 9.1.3 SP4 on Windows, the maximum length of the macro text expression is 65534 characters. Any text expression that is longer than the limit can cause an error. For example, we can create a 65534 character long value like:

```
%let long =%sysfunc(repeat(X,65534-1));
```

Any more characters will cause an error. For instance, the %EVAL below sees an expression of 65535 character long, including the equal sign(=):

```
%put %eval(&long=); /* error */
```

One character less, then the %EVAL is happy.

```
%let longMinus=%sysfunc(repeat(X,65534-1-1));
%put %eval(&longMinus=);          /* 0 */
```

Quoted, however, %EVAL is unhappy again. This time, we forgot to count the invisible pair of delta characters at both ends of the quoted string:

```
%put %eval(%superq(longMinus)=); /* error */

%let longMinus3=%substr(&long,4);
%put %eval(%superq(longMinus3)=); /* 0 */
```

Macro system, however, works with a token, not necessarily with the whole text expression all the time. Sometimes a macro text expression that looks like impossibly long can cause no problem. This is because parts of the text expression are resolved to shorter ones, so that when the whole thing is re-scanned, it becomes manageable. For instance, the following works fine, despite the appearance of the implicit %EVAL receiving two very long parameter values:

```
%let longMinus3=%sysfunc(repeat(X,65534-1-3));
%macro test2;
  %if %isBlank(&longMinus3) and %isBlank(&longMinus3) %then %put both blank;
  %else %put at least either is not blank;
%mend  test2;
%test2
```

Our %ISBLANK macro (i.e., C8) can take a parameter up to 65531 character long, which is just three character less than the maximum. That is, assuming that not all of the 65531 characters are numeric digits. In fact, hundreds of digits are fine, but thousands of digits seem to hang the session.[2]

Other Variations

Finally, we discuss two other possible “contenders” of the Is-Blank (con)test. The first one is:

```
%if %length(%superq(param)) = 0 %then ...
```

Above will fail with the inputs involving quoted blanks (%STR( )), because this is a test of Is-Null, not of Is-Blank. A more interesting case is:

```
%if %superq(param) eq %str() %then ...
```

[2] This seems to happen only with SAS on the 32-bit Windows platform. A recent version of SAS session running on a 64-bit Windows platform does not hang when the %SYSEVALF() encounters thousands of digits. SAS on Unix or z/OS platforms does not seem to hang, either.

which has no problem finding the quoted blanks as "blank." This may be a surprise to those who recall that %EVAL strips off the leading and trailing blanks from a string that is compile-time quoted (that is, quoted by either %STR or %NRSTR). But %EVAL does not strip if the blanks are execution-time quoted (that is, quoted by %BQUOTE or %NRBQUOTE): For example:

```
%put %eval(     %str(  A)=A); /* 1 */
%put %eval(   %nrstr(  A)=A); /* 1 */
%put %eval(  %bquote(  A)=A); /* 0 */
%put %eval(%nrbquote(  A)=A); /* 0 */
```

The above %IF test however works since %EVAL compares blanks equal to blanks no matter how many there are on either side or whether they are macro quoted or not. So, the real reason why we do not bother to consider the above %IF test is that the right side of the equal to operator (i.e., %STR()) does not do much in the context. If anything, it rather confuses users, because it seems to suggest only quoted nothing should be compared true, but in fact, any number of quoted blanks will do fine, as well.

SUMMARY

After reviewing eight different ways to test if a given parameter is "blank," we recommend a simple test of equality against nothing after quoting the input parameter reference with the powerful macro quoting function, %SUPERQ(), in the context of %SYSEVALF. We also provide this recommendation with a single-line macro, %ISBLANK(PARAM). The test results are discussed in detail, hoping that the discussion leads to a deeper understanding of many issues in implementing macro parameter tests.

REFERENCES

Data _Null_ (2008). "Macro variable resolves to – question" A SAS-L posting on April 7, 2008. Archived at < http://groups.google.com/group/comp.soft-sys.sas/msg/6c7402de338de9f9>.

Beilei Xu and Lei Zhang. (2004). "Take and In-Depth Look at the %EVAL Function." Paper pm26. In *The Proceedings of the North East SAS Users Group Conference* (NESUG2004). Available at < http://www.nesug.info/Proceedings/nesug04/pm/pm26.pdf>.

SAS Institute Inc. (2007). "Sample 25013: Power and sample size for independence tests in RxC tables." Published as a Sample <http://support.sas.com/kb/25/013.html>. Source downloaded from <http://support.sas.com/kb/25/addl/fusion25013_1_powerrxc.sas.txt> on April 22, 2008.

SAS Institute Inc. (2008). *SAS 9.2 Macro Language: Reference.* ISBN: 978-1-59047-975-9. Cary, NC, USA. Also available (as of Dec. 2008) in PDF and HTML format at <http://support.sas.com/documentation/onlinedoc/base/index.html>.

Scott Bucher. (2008) "checking for missing parameters" A SAS-L posting on Jan 31, 2008. Archived at < http://www.listserv.uga.edu/cgi-bin/wa?A2=ind0801E&L=sas-l&D=1&P=34885>.

ACKNOWLEDGEMENTS

Our thanks to the sas-l online discussion group, which is an important part of our professional *and* personal life. This paper is a direct follow-up of data _null_'s question posted on sas-l (2008) and an earlier one by Scott Bucher (2008). We thank those who contributed on the discussion threads: Toby Dunn, Ronald J. Fehd, Ed Heaton, Don Henderson, Gerhand Hellriegel, Tom Hide, Chris Long, Stefan Pohl, Roland Rashleigh-Berry, and Ian Whitlock. Our special thanks to Jason Secosky at SAS Institute Inc., who kindly found time to read a draft and made great suggestions that improved this paper significantly.

CONTACT INFORMATION

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

John King
Ouachita Clinical Data Services, Inc.
24 Loblolly Circle
Mount Ida, AR 71957

APPENDIX. SAS CODE FOR TESTING

```
/* isBlankTest.sas
   purpose: to automatize the test of various implementation of isBlank
            macro ("contenders") with various input parameter values.
            this is an appendix of the conference paper titled "Is
            this macro parameter blank?"
   input  : nothing
   output : given v parameter values and c contenders, it will
            write out v * c sas source code files (.sas) and
            a log file (.log) for each, in the isBlankTest sub-
            directory under the WORK directory (which will be created
            if it does not already exist)
   note   : ran with SAS 9.1.3 SP4 for Windows. no guarantees what so
            ever, comments are welcome.
   ver    : 0.4 (2008-10-09)
   by     : chang y chung and john king
   contact: chang_y_chung@hotmail.com

*/

options noovp nocenter;

*cd to work directory ------------------------------------------------;;
x cd %sysfunc(pathname(WORK));

*files and directories -----------------------------------------------;;
filename test "isBlankTest"; *test programs will be written here;
filename exec %sysfunc(quote(%sysfunc(sysget(sasroot))\sas.exe));
filename incl temp; *for %include;

*input parameter values ----------------------------------------------;;
data vs;
  infile cards truncover firstobs=2;
  input vid $ blank $ v $ 6-40;
  /* (nothing) will be converted to nothing later */
cards;
----+----1----+----2----+----3----+----4
01 1 (nothing)
02 1 %str( )
03 0 &
04 0 A>B+C>D
05 0 %bquote(Tom&Jerry)
06 0 %nrstr(%()
07 0 1 or
08 0 %sysfunc(repeat(X,65533-1))
09 1 %qsysfunc(repeat(%str( ),10000-1))
10 0 43876106244387610624
;
run;

*contenders ----------------------------------------------------------;;
data cs;
  infile cards truncover firstobs=2;
  input cid $ c $ 3-40;
cards;
```

```
----+----1----+----2----+----3----+----4
01 &param eq
02 %bquote(&param)=
03 %nrbquote(&param)=
04 %superq(param)=
05 "&param" = ""
06 %length(&param) = 0
07 %length(%qleft(%qtrim(&param))) = 0
08 %sysevalf(%superq(param)=,boolean)
;
run;

*cartesian product. also add a var, vc ------------------------------;;
data vcs;
  set vs;
  do p = 1 to n;
    set cs point=p nobs=n;
    vc = catt("v", vid, "c", cid);
    output;
  end;
run;

*test macro code template ------------------------------------------;;
data tmpl;
  infile cards truncover firstobs=2;
  input line $char55.;
cards4;
----+----1----+----2----+----3----+----4----+----5----+
/* test macro vid=[vid] cid=[vid]*/
options ls=max ps=max;
%macro test(param);
  %put &prefix IN entry=Success;
  %if [c] %then %let res=1;
  %else %let res=0;
%mend  test;

/* prepare for the invocation */
%let prefix = NOTE: (%str(TEST));
%put &prefix WHAT vid=[vid] cid=[cid];
%let start = %sysfunc(time());
%let res=;

/* macro call */
%test([v])

/* output results */
%put &prefix RES
  res=%sysfunc(ifc(&res=[blank],Correct,Incorrect))
  time=%sysevalf(%sysfunc(time())-&start);
;;;;
run;

*create the test directory or clean up the existing one --------------;;
data _null_;
  length testpath filepath dn $256 fn $8;
  testpath = pathname("test");
  if not fileexist(testpath) then do;
    dn = dcreate(scan(testpath, -1, "\/"));
    if missing(dn) then error "ERROR: cannot create dir " testpath;
```

```
  end;
  did = dopen("test");
  if did = 0 then error "ERROR: cannot open dir " testpath;
  dnum = dnum(did);
  if not missing(dnum) then do i = 1 to dnum;
    filepath = catx("/", testpath, dread(did, i));
    rc = filename(fn, filepath);
    rc = fdelete(fn);
    if rc^ = 0 then error "ERROR: cannot delete file " filepath;
    rc = filename(fn,"");
  end;
  rc = dclose(did);
run;

*write out test sas files and test driver to be included -------------;;
data _null_;

  set vcs;

  length testpath fname line $256;
  testpath = pathname("test");
  fname = catx("\", testpath, catx(".", vc, "sas"));

  file dummy filevar=fname; *test sas files;
  do p = 1 to n;
    set tmpl point=p nobs=n;
    line = tranwrd(line, "[vid]", trimn(vid));
    line = tranwrd(line, "[cid]", trimn(cid));
    line = tranwrd(line, "[v]", trimn(ifc(v="(nothing)","",v)));
    line = tranwrd(line, "[c]", trimn(c));
    line = tranwrd(line, "[blank]", trimn(blank));
    len = length(line);
    put line $varying. len;
  end;

  file incl; *test driver to be included;
  length exec opts sysin log cmd $256;
  exec = quote(trimn(pathname("exec")));
  opts = "-nocenter -noovp -nosplash -rsasuser";
  sysin = catx(" ", "-sysin", quote(trimn(fname)));
  log = catx(" ", "-log", quote(trimn(testpath)));
  cmd = quote(catx(" ", exec, opts, sysin, log));
  len = length(cmd);
  status = vc;
  put "systask command" +1 cmd $varying. len +1 status= +1 "wait;";
  put '%put NOTE: ' status '=&' status ';';
run;

*run all test sas files ----------------------------------------------;;
%include incl/ source2;

*read the log files into an analysis dataset -------------------------;;
data results;
  length vid cid $2 entry $7 res $9;
  retain vid cid entry res " " warned erred time .;
  infile test('*.log') eov=eov eof=eof truncover;
  input @1 msgType $ @1 @("NOTE: (TEST)") noteType $ @;
  if eov then goto eof;
  select(msgType);
```

```
    when("ERROR:") if entry="Success" & missing(res) then erred = 1;
    when("WARNING:") if entry="Success" & missing(res) then warned = 1;
    when("NOTE:") do;
      select(noteType);
        when("WHAT") input vid= cid=;
        when("IN") input entry=;
        when("RES") input res= time=;
        otherwise;
      end;
    end;
    otherwise;
  end;
  input;
  return;
eof:
  keep vid--time;
  output;
  eov = 0;
  call missing(entry, res, warned, erred, time);
run;

*report test results ----------------------------------------------------;;
proc print data=results noobs;
  by vid;
run;

/* on lst
vid    cid     entry        res       warned    erred      time

01     01     Success    Correct         .         .      0.00100
01     02     Success    Correct         .         .      0.00100
01     03     Success    Correct         .         .      0.00100
01     04     Success    Correct         .         .      0.00100
01     05     Success    Correct         .         .      0.00100
01     06     Success    Correct         .         .      0.00100
01     07     Success    Correct         .         .      0.07100
01     08     Success    Correct         .         .      0.00100
02     01     Success    Correct         .         .      0.00100
02     02     Success    Correct         .         .      0.00100
02     03     Success    Correct         .         .      0.00100
02     04     Success    Correct         .         .      0.00100
02     05     Success    Incorrect       .         .      0.00100
02     06     Success    Incorrect       .         .      0.00100
02     07     Success    Correct         .         .      0.05300
02     08     Success    Correct         .         .      0.00100
03     01     Success    Incorrect       .         1      0.00100
03     02     Success    Incorrect       .         1      0.00100
03     03     Success    Correct         .         .      0.00100
03     04     Success    Correct         .         .      0.00100
03     05     Success    Correct         .         .      0.00100
03     06     Success    Correct         .         .      0.00100
03     07     Success    Correct         .         .      0.01000
03     08     Success    Correct         .         .      0.00000
04     01     Success    Incorrect       .         1      0.00100
04     02     Success    Correct         .         .      0.00100
04     03     Success    Correct         .         .      0.00000
04     04     Success    Correct         .         .      0.00000
04     05     Success    Correct         .         .      0.00100
04     06     Success    Correct         .         .      0.00100
```

```
04      07      Success     Correct          .          .       0.06100
04      08      Success     Correct          .          .       0.00100
05      01      Success     Incorrect        .          1       0.00100
05      02      Success     Incorrect        1          1       0.00100
05      03      Success     Correct          .          .       0.00100
05      04      Success     Correct          .          .       0.00100
05      05      Success     Correct          .          .       0.00100
05      06      Success     Correct          .          .       0.00200
05      07      Success     Correct          1          .       0.00800
05      08      Success     Correct          .          .       0.00000
06      01      Success     Correct          .          .       0.00100
06      02      Success     Correct          .          .       0.00100
06      03      Success     Correct          .          .       0.00100
06      04      Success     Correct          .          .       0.00000
06      05      Success     Correct          .          .       0.00000
06      06      Success     Correct          .          .       0.00100
06      07      Success     Correct          .          .       0.00900
06      08      Success     Correct          .          .       0.00100
07      01      Success     Incorrect        .          .       0.00000
07      02      Success     Correct          .          .       0.00100
07      03      Success     Correct          .          .       0.00100
07      04      Success     Correct          .          .       0.00100
07      05      Success     Correct          .          .       0.00100
07      06      Success     Correct          .          .       0.00200
07      07      Success     Correct          .          .       0.00900
07      08      Success     Correct          .          .       0.00100
08      01      Success     Correct          .          .       0.04700
08      02      Success     Correct          .          .       0.04500
08      03      Success     Correct          .          .       0.07000
08      04      Success     Correct          .          .       0.06500
08      05      Success     Correct          .          .       0.03100
08      06      Success     Correct          .          .       0.06300
08      07      Success     Incorrect        .          1       0.11100
08      08      Success     Correct          .          .       0.03900
09      01      Success     Correct          .          .       0.03300
09      02      Success     Correct          .          .       0.03500
09      03      Success     Correct          .          .       0.03500
09      04      Success     Correct          .          .       0.04200
09      05      Success     Incorrect        .          .       0.03400
09      06      Success     Incorrect        .          .       0.04200
09      07      Success     Correct          .          .       3.63400
09      08      Success     Correct          .          .       0.04200
10      01      Success     Incorrect        .          1       0.00100
10      02      Success     Incorrect        .          1       0.00100
10      03      Success     Incorrect        .          1       0.00100
10      04      Success     Incorrect        .          1       0.00100
10      05      Success     Correct          .          .       0.00100
10      06      Success     Correct          .          .       0.00200
10      07      Success     Correct          .          .       0.04500
10      08      Success     Correct          .          .       0.00100
*/
```