Paper AD-103

%Destroy() a Macro With Permutations

Brandon Welch, Rho®, Inc., Chapel Hill, NC
James Vaughan, Rho, Inc., Chapel Hill, NC

ABSTRACT

The SAS® Macro is a powerful tool. It minimizes repetitive tasks and provides portable tools for users. When these tools are delivered to clients we want them to be of the highest quality. For example, when developed to perform a complicated statistical test, the macro should produce accurate results and a clean log. To accomplish this, we insert parameter checks. Depending on the complexity of the macro it is sometimes difficult to perform a thorough check. We introduce the `%Destroy()` macro which uses the `SYSCALL RANPERK` routine to permute a list of arguments. These arguments are then passed to the macro to test. We show how to add appropriate parameter checks, ensuring on subsequent runs of `%Destroy()` the testing macro produces the desired results. While this article targets a clinical computing audience, the techniques we present offer a good overview of macro processing that will educate SAS programmers of all levels across various disciplines.

INTRODUCTION

In a statistical computing environment, we often create macros to perform repetitive or complicated tasks. For example, a statistician may need to perform an exotic statistical task and place the code in a macro. This creates a portable tool that a programmer can use without having to know the exact details of how the statistical test works. It is good practice to have the macro validated and for the macro to produce a clean log when called. In this paper we will only focus on producing a clean log. To do this we introduce the `%Destroy()` macro. This macro calls a macro and passes a series of 'good' and 'bad' arguments at random. For the purposes of this paper, the macro we are testing with `%Destroy()` will be called the 'testing macro'. In the case where the argument is 'bad' or causes the testing macro to fail, we add parameter checks. These checks ensure that on subsequent calls to the testing macro, we get a clean log. Our goal is to add parameter checks to our testing macro such that when `%Destroy()` is invoked, we have a clean log.

EXAMPLE

Here's a simple macro (`Contin`) that computes a mean. We create two data sets (DAT1 and DAT2) with variables VAR1, VAR2 and TRT. With the arguments used below, it works fine and exits gracefully:

```
DATA dat1;
  do i = 1 to 30;
    var1  = rangam(12345,10);
    trt   = ranbin(12345,3,0.3) + 1;
    output;
  end;
RUN;

DATA dat2;
  do i = 1 to 30;
    var2   = rannor(12345,10);
    trt    = ranbin(12345,3,0.3) + 1;
    output;
  end;
RUN;

%macro Contin(InDat  = , Sub = , Value  =);

   DATA anly;
     set &InDat.(where = (trt = &Sub.));
   RUN;

   PROC MEANS data = anly;
     var &Value;
```

```
      RUN;

   %mend;

   %Contin(InDat = dat1, Sub   = 3, Value   = var1);
```

Now suppose we pass a 'bad' argument to the `InDat=` parameter:

```
   %Contin(InDat = badarg, Sub   = 3, Value  = var1);
```

Since the data set 'badarg' does not exist we get:

```
   ERROR: File WORK.BADARG.DATA does not exist.
```

To get around this we can add the following to `%Contin()` right before the first data step:

```
   %if %sysfunc(exist(&InDat)) = 0 %then %do;
     %put -------- DATA SET (&InDat) DOES NOT EXIST, MACRO Contin STOPPING;
     %return;
   %end;
```

The macro will stop before it gets to first data step and no ERRORS are issued to the log. Instead, we get:

```
-------- DATA SET (badarg) DOES NOT EXIST, MACRO Contin STOPPING
```

This is a good example of a parameter check that stops execution when we have no data. This is only one example. By using the `%Destroy()` macro we develop a list of 'good' and 'bad' arguments and pass them all at random, so more cases are tested. For those cases that produce WARNINGS and/or ERRORS we can implement the appropriate parameter check to insure the log is clean. After we add all our parameter checks, any call to the testing macro from `%Destroy()` will result in a clean log. Before we build `%Destroy()` permutations are discussed.

PERMUTATIONS

In order to sufficiently test the macro's ability to identify and correctly handle both 'good' and 'bad' arguments, a list of possible arguments ('good' and 'bad') is compiled for each macro parameter. Since an argument may be considered 'good' for one parameter, but 'bad' for another, we expand the list of possible argument sets by randomly permuting (or rearranging) the values from the full set of arguments of size n into a set of size k, with particular sequence or order without repetition, also known as sequences without repetition.

For the example macro (`Contin`) there are *x* arrangements (or sets) of a fixed length of 3 arguments ('good' and/or 'bad') taken from a list of size n where $x = {}_nP_3$ (or the number of permutations of n items that are taken 3 at a time).

The total number of permutations generated is set to ${}_nP_3$; however, because the values are randomly permutated, not all possible permutations of k out of n items may be generated.

BUILDING `%DESTROY()`

We use the `%Contin()` macro as our testing macro. We first create a list of arguments resulting in 'good' and 'bad' cases. These will be randomly passed to `%Contin()`. We chose "`var1 3 a var2 dat1 dat2`". Note that some permutations of these yield desirable results (like above, `dat1 3 var1`) and some do not.

`%Destroy()` has 4 parameters:

`Mac` = Name of the testing macro (`%Contin()` in our example)
`Seed` = Seed used for `SYSCALL RANPERK`
`TestArgs` = List of test arguments, separated by vertical bar (`var1|3|a|var2|dat1|dat2`)
`ParmNum` = Number of parameters in testing macro, `%Contin()` has three parameters

Step 1: Count the number of test arguments (six in our example):

```
   %let TargCnt = %sysfunc(countw(&TestArgs.,|));
```

Step 2: Determine total number of permutations (${}_6P_3 = 120$):

```
%let PermNum = %sysfunc(perm(&TArgCnt.,&ParmNum.));
```

Step 3: Split list of arguments into separate arguments and make comma delimited (required for `SYSCALL RANPERK`)

```
%let Args = ;
%do i = 1 %to &TArgCnt;
  %let p&i = %scan(&TestArgs.,&i,|);
  %if &i = 1 %then %let Args = p&i;
  %else            %let Args = &Args%str(, )p&i;
%end;
```

Step4: Generate permutations and call testing macro for each permutation:

```
%do j = 1 %to &PermNum;

  %*GENERATE PERMUTATIONS;
  %syscall ranperk(seed, ParmNum, %unquote(&Args));

  %put -----------;
  %put &p1 &p2 &p3;
  %put -----------;

  %*CALL MACRO FOR EVERY PERMUTATION;
  %&Mac.(InDat = &p1,
         Sub   = &p2,
         Value = &p3);
%end;
```

Note for `%syscall`, we must mask the `&` for each `&pi`. If we do not mask the `&`, the explicit variable values will resolve within `RANPERK` – subsequently throwing an error.

Partial text in log from `%put`:

```
-----------
a dat1 dat2
-----------
dat1 a 3
-----------
dat1 dat2 a
-----------
3 var2 a
-----------
.
.
.
```

IMPLEMENTING %DESTROY() AND ADDING PARAMETER CHECKS

Using `%Contin()` as our testing macro, after running `%Destroy()` we get:

Message Type	Message
Error	File WORK.A.DATA does not exist.
Error	File WORK.VAR1.DATA does not exist.
Error	File WORK.VAR2.DATA does not exist.
Error	Syntax error, expecting one of the following
Error	Syntax error, statement will be ignored.
Error	The symbol is not recognized and will be ignored.

Message Type	Message
Error	Variable A not found.
Error	Variable DAT1 not found.
Error	Variable DAT2 not found.
Error	Variable VAR1 not found.
Error	Variable VAR2 not found.
Error	Variable a is not on file WORK.DAT1.
Error	Variable a is not on file WORK.DAT2.
Error	Variable dat1 is not on file WORK.DAT2.
Error	Variable dat2 is not on file WORK.DAT1.
Error	Variable var1 is not on file WORK.DAT2.
Error	Variable var2 is not on file WORK.DAT1.
Warning	Data set WORK.ANLY was not replaced because this step was stopped.
Warning	The data set WORK.ANLY may be incomplete. When this step was stopped there were 0

Table 1. First log report

Our first priority is to ensure the macro stops if our input data set does not exist. We saw above that by using `%sysfunc(exist())`, we can avoid this error. In addition, a data set may exist but not have any observations. We use the automatic variable `&SQLObs` to determine if a data set has zero observations. If this occurs, the macro stops. Note we assign the value of `&SQLObs` to `&DatObs`. We reset the value to null at each call to `%Contin()`. This prevents values from previous calls to `%Contin()` from carrying over to the current call.

```
PROC SQL noprint;
  select * from anly;
QUIT;
%let DatObs = &SQLObs;

%if &DatObs = 0 %then %do;
  %put -------- DATA SET (&InDat) HAS ZERO RECORDS, MACRO Contin STOPPING;
  %return;
%end;
```

After adding these checks, we get:

Message Type	Message
Error	Syntax error, expecting one of the following
Error	The symbol is not recognized and will be ignored.
Error	Variable A not found.
Error	Variable DAT1 not found.
Error	Variable DAT2 not found.
Error	Variable VAR1 not found.
Error	Variable VAR2 not found.
Error	Variable a is not on file WORK.DAT1.
Error	Variable a is not on file WORK.DAT2.
Error	Variable dat1 is not on file WORK.DAT2.
Error	Variable dat2 is not on file WORK.DAT1.

Message Type	Message
Error	Variable var1 is not on file WORK.DAT2.
Error	Variable var2 is not on file WORK.DAT1.
Warning	Data set WORK.ANLY was not replaced because this step was stopped.
Warning	The data set WORK.ANLY may be incomplete. When this step was stopped there were 0

Table 2. Second log report

Now that the explicit data issues are resolved, we focus on the variables. One of the arguments is a number (3). Variable names cannot be numbers. So we add:

```
%if %datatyp(&Value) ne CHAR %then %do;
  %put -------- VARIABLE NAME (&Value) IS A NUMBER IN &InDat, MACRO Contin
  STOPPING;
  %return;
%end;
```

Also, for our WHERE expression in the ANLY data step, the TRT variable must be numeric. We use the `%datatyp()` autocall macro again:

```
%if %datatyp(&Sub) = CHAR %then %do;
  %put -------- TREATMENT VARIABLE VALUE (&Sub) IS INVALID (MUST BE
  NUMERIC), MACRO Contin STOPPING;
  %return;
%end;
```

After adding these checks to `%Contin()`, we get:

Message Type	Message
Error	Variable A not found.
Error	Variable DAT1 not found.
Error	Variable DAT2 not found.

Table 3. Final log report

All that's left now is to check for variable existence. We do this by using I/O functions.

```
%let DSId  = %sysfunc(open(&InDat));
%let DSObs = %sysfunc(attrn(&DSId,NOBS));

%if &DSObs > 0 %then %do;
  %if %sysfunc(VARNUM(&DSId,&Value)) = 0 %then %do;
    %put -------- VARIABLE (&Value) DOES NOT EXIST IN &InDat, MACRO Contin
    STOPPING;
    %return;
  %end;
%end;

%let RC = %sysfunc(close(&DSId));
```

CONCLUSION

It is always good practice to maintain a clean log. This is especially important when we delivery programs to our clients. If our programs use macros for repetitive tasks or complicated statistical tests, the `%Destroy()` macro offers a good solution to ensure the macro works properly and produces a clean log. This paper introduces some advanced macro techniques. But equally important, it shows the reader some tools that determine we have valid arguments (e.g. using `%sysfunc(exist())` when determining a data set's existence). What we show in this paper are only a

few examples. By feeding macro ‘good’ and ‘bad’ arguments at random, one can test many types of cases and ultimately produce a superior product.

ACKNOWLEDGMENTS

Eva J. Welch
Dave Scocca
Steve Noga

CONTACT INFORMATION

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

Brandon Welch
Rho, Inc
6330 Quadrangle Dr., Ste. 500
Chapel Hill, NC 27517
Phone: 919-595-6592
Fax: 919-408-0999
Email: Brandon_Welch@rhoworld.com
Web: www.rhoworld.com

APPENDIX

```
%macro Destroy(Mac      = ,
               Seed     = ,
               TestArgs = ,
               ParmNum   =
               );

  %*COUNT THE NUMBER OF TEST ARGUMENTS;
  %let TargCnt = %sysfunc(countw(&TestArgs.,|));
  %put ---------------------------------------;
  %put -----NUMBER OF TEST ARGUMENTS: &TArgCnt;
  %put ---------------------------------------;

  %*GET NUMBER OF PERMUTATIONS TO SCROLL THROUGH;
  %let PermNum = %sysfunc(perm(&TArgCnt.,&ParmNum.));
  %put ---------------------------------------;
  %put -----NUMBER OF PERMUTATIONS: &PermNum;
  %put ---------------------------------------;

  %*SPLIT TEST ARGUMENT STRING INTO SEPARATE ARGUMENTS;
  %let Args = ;
  %do i = 1 %to &TArgCnt;
    %let p&i = %scan(&TestArgs.,&i,|);
    %if &i = 1 %then %let Args = p&i;
    %else            %let Args = &Args%str(, )p&i;
  %end;

  %do j = 1 %to &PermNum;

    %*GENERATE PERMUTATIONS;
    %syscall ranperk(seed, ParmNum, %unquote(&Args));

    %put -----------;
    %put &p1 &p2 &p3;
    %put -----------;

    %*CALL MACRO FOR EVERY PERMUTATION;
    %&Mac.(InDat = &p1,
           Sub   = &p2,
           Value = &p3);
  %end;

%mend;

/*******CONTIN MACRO INCLUDING ALL PARAMETER CHECKS*******/

%macro Contin(InDat = ,
              Sub   = ,
              Value =);

  %let DatObs = ;

  %if %sysfunc(exist(&InDat)) = 0 %then %do;
    %put -------- DATA SET (&InDat) DOES NOT EXIST, MACRO Contin STOPPING;
    %return;
  %end;

  %if %datatyp(&Value) ne CHAR %then %do;
    %put -------- VARIABLE NAME (&Value) IS A NUMBER IN &InDat, MACRO Contin
    STOPPING;
    %return;
  %end;

  %if %datatyp(&Sub) = CHAR %then %do;
    %put -------- TREATMENT VARIABLE VALUE (&Sub) IS INVALID (MUST BE
```

```
      NUMERIC), MACRO Contin STOPPING;
      %return;
    %end;

    %let DSId  = %sysfunc(open(&InDat));
    %let DSObs = %sysfunc(attrn(&DSId,NOBS));

    %if &DSObs > 0 %then %do;
      %if %sysfunc(VARNUM(&DSId,&Value)) = 0 %then %do;
        %put -------- VARIABLE (&Value) DOES NOT EXIST IN &InDat, MACRO Contin
        STOPPING;
        %return;
      %end;
    %end;

    %let RC = %sysfunc(close(&DSId));

    DATA anly;
      set &InDat.(where = (trt = &Sub.));
    RUN;

    PROC SQL noprint;
      select * from anly;
    QUIT;
    %let DatObs = &SQLObs;

    %if &DatObs = 0 %then %do;
      %put -------- DATA SET (&InDat) HAS ZERO RECORDS, MACRO Contin STOPPING;
      %return;
    %end;

    PROC MEANS data = anly;
      var &Value;
    RUN;

  %mend;
```