SAS MACRO — AN INTRODUCTION
John J. Genzano, III
Genzano Software Consulting

Many times, in the course of solving a problem, we have found need to reuse some of our existing SAS code. In order to do this, we have found ourselves editing the existing program to handle the new, yet similar, situation. In other cases, we have found ourselves duplicating code many times in a single program, making the minor changes needed in each code segment to handle the minor differences in the situations that caused us to copy that code. The SAS Macro language has many of the capabilities needed to rectify the above situations.

In reality, there is very little chance that I can teach you even an introduction to the SAS Macro language in an hour, or in a paper this short. What I can do, however, is give you an understanding of how the SAS Macro language works, so that you can better understand the SAS Macro manual, and will have fewer problems in using the SAS Macro language.

In order to do that, an understanding of how SAS itself works is needed. Computer languages, in general, come in three types; those that are interpreted, those that are compiled, and those that are a hybrid of the other two. An interpreted language is one where each individual line of source code is “interpreted” into machine language and executed as it is read. Probably the most familiar example of this is the PC batch language, the language that interprets your AUTOEXEC.BAT. The advantage of this kind of language is that it is easy to develop and even easier to fix errors, as it is the actual source code that is executed, there are no steps involved beyond writing the source code in developing a program. The major problem with this kind of language is speed; since each line of code must be read and interpreted as it is executed, these interpreted languages tend to be slow.

Compiled languages are those where the source code is converted, as a whole, to machine language and the machine language version is then saved and executed directly. The advantage to this method is speed, since the program is already in “machine language” at execution time, there is no further translation necessary. The major disadvantages are in development and problem fixing (or “debugging”). Once the source code is written, it must go through a second (and sometimes third) process to convert it into machine language before it can be run. Then, if a problem or error occurs, there is no longer a true “link” between the executable program and the source code, so fixing those problems can be very difficult. Because of this, many tools and techniques have been developed to enhance the debugging capabilities of many compiled languages. The best examples of compiled languages are COBOL or FORTRAN or C.

The third type of program, the hybrid, generally comes in a type where the source code is “compiled” to a form that is then interpreted by the computer. An older example of this is the original PASCAL, which “compiled” into what was called “P-code” and that P-code file was then saved. When you ran your “compiled” program, the P-code was actually interpreted at run time. Visual Basic works this way, which is why you need the VBRUN300.DLL program to run a Visual Basic version 3 program. That is the “interpreter” for the Visual Basic “compiled” language.

SAS is also a hybrid language, but a different type than the hybrids mentioned above. A SAS program file is stored as source code, but each “run group” in the file is compiled and executed as it is read from the file. This allows for fairly easy debugging, especially in the “window” environment, as the section of code with the error is identified easily, yet each section of code is compiled, adding speed to the actual run times.

What does this mean to you, in terms of the SAS Macro language? Well, it helps explain how the language works. Let’s examine this absolutely contrived hypothetical example. In a school, each class maintains grade records in its own SAS dataset. The dataset has two fields, STDNT_ID (Student Identifier) and TST_SCOR (Test Score), with one record for each test the student took in that class. At the end of the year, a program needs to be run for all the classes to accumulate grade averages for each student. One way to do this follows:

Example 1

DATA ENGLISH;
SET PERM.ENGLISH;
BY STDNT_ID;
RETAIN TST_CNT TOT_SCOR;
IF FIRST.STDNT_ID THEN DO;
  TST_CNT = 0;
  TOT_SCOR = 0;
END;
TST_CNT = TST_CNT + 1;
TOT_SCOR = TOT_SCOR + TST_SCOR;
IF LAST.STDNT_ID THEN DO;
  AVG_SCOR = TOT_SCOR / TST_CNT;
OUTPUT;
END;
KEEP STDNT_ID AVG_SCOR;
DATA HISTORY;
BEGINNING TUTORIALS

SET PERM.HISTORY;
BY STDNT_ID;
RETAIN TST_CNT TOT_SCOR;
IF FIRST.STDNT_ID THEN DO;
  TST_CNT = 0;
  TOT_SCOR = 0;
END;
TST_CNT = TST_CNT + 1;
TOT_SCOR = TOT_SCOR + TST_SCOR;
IF LAST.STDNT_ID THEN DO;
  AVG_SCOR = TOT_SCOR / TST_CNT;
  OUTPUT;
END;
KEEP STDNT_ID AVG_SCOR;

And then keep repeating this code for each class the
school offers. SAS Macro allows us to do it in a much
easier way, as follows:

Example 2

%MACRO AVGSCORE;
RETAIN TST_CNT TOT_SCOR;
IF FIRST.STDNT_ID THEN DO;
  TST_CNT = 0;
  TOT_SCOR = 0;
END;
TST_CNT = TST_CNT + 1;
TOT_SCOR = TOT_SCOR + TST_SCOR;
IF LAST.STDNT_ID THEN DO;
  AVG_SCOR = TOT_SCOR / TST_CNT;
  OUTPUT;
END;
KEEP STDNT_ID AVG_SCOR;
%MEND;

DATA ENGLISH;
  SET PERM.ENGLISH;
  BY STDNT_ID;
  %AVGSCORE

DATA HISTORY;
  SET PERM.HISTORY;
  BY STDNT_ID;
  %AVGSCORE

%LET CLASS = ENGLISH;
%AVGSCORE

And then just repeating that smaller piece of code for each
class the school offers. What happens in the situation
above is EXACTLY the same as in the original code. The
reason is that the SAS Macro language is a "substitution"
language. To explain this, let us examine the second set of
code, above. The first thing we see is a command
%MACRO AVGSCORE. The percent sign in the front
tells SAS that the command is going to the SAS Macro
processor. The command tells the SAS Macro processor
to start defining a "macro" named AVGSCORE. The SAS
Macro processor then stores all of the source code into
that macro until it gets to the %MEND command, which
tells it that it has reached the end of the macro. SAS then
continues reading the program source code and starts a
DATA step. It sets up all the required buffers and data
areas, then it gets to the %AVGSCORE command. Since
the command starts with a percent sign, SAS gives the
command to the Macro processor. The Macro processor
examines it, notices it is not a "built-in" macro command,
and so assumes it is the name of a macro and goes looking
for a macro with that name. When it finds it, it takes the
source code stored in that location and puts it into the
code, replacing the %AVGSCORE command. (That is
why the %AVGSCORE command is not followed by a
semi-colon, as one would expect. All of the required semi­
colons are already in the macro code.) This is also known
as "calling" the macro, or the macro is "called" when SAS
gets to the %AVGSCORE command, and the macro
processor makes the substitution. So, by substituting the
SAS code that is between the %MACRO AVGSCORE
and %MEND commands into the DATA steps, you get
EXACTLY the same code as in example 1. All this
substitution occurs before SAS compiles the data step.

Now then, if this was all that the SAS Macro language
could do, while being useful, it would be pretty limited.
But the SAS Macro language is not limited at all. One of
the major strengths of the language is the ability to use
substitution variables, also called "AMPER" variables
because they are designated by an ampersand (&) in front.
Let's examine another set of code to accomplish our
school system that makes use of ampers variables.

Example 3

%MACRO AVGSCORE;
DATA &CLASS;
  SET PERM.&CLASS;
  BY STDNT_ID;
  %AVGSCORE

DATA ENGLISH;
  SET PERM.ENGLISH;
  BY STDNT_ID;
  %AVGSCORE

And then just repeating that smaller piece of code for each
class the school offers. What happens in the situation
above is EXACTLY the same as in the original code. The
reason is that the SAS Macro language is a "substitution"
language. To explain this, let us examine the second set of
code, above. The first thing we see is a command
%MACRO AVGSCORE. The percent sign in the front
tells SAS that the command is going to the SAS Macro
And then repeating just those two lines for each class the school offers. How does this macro work? It produces EXACTLY the same code as Example 1. Let's take a look and see how.

Again, the SAS Macro processor starts storing the source code for the macro away when it sees the %MACRO command, just as in Example 2. The next thing that happens is that SAS sees a %LET command, so it again calls the SAS Macro processor to handle it (it begins with a percent sign, so it must be a MACRO command). The macro processor sets aside a variable space, labels it "CLASS" and stores the value ENGLISH in that space. Now SAS sees the %AVGSCORE command and again calls the macro processor, who does exactly what it did in Example 2, and substitutes the code stored in the AVGSCORE macro into the code stream, where SAS starts to process it. SAS starts to set up a DATA step, but runs into the &CLASS token. This causes SAS to look in the variable space set up by the macro processor for a space labeled CLASS, and it substitutes whatever it finds there for the token &CLASS. So, in the first case, the substitution produces DATA ENGLISH; and in the second case, the substitution produces DATA HISTORY; thereby producing EXACTLY the same code as in Example 1. An even shorter method for producing the same effect as in Example 3 follows:

Example 4

%MACRO AVGSCORE(CLASS=);  
DATA &CLASS;  
SET PERM.&CLASS;  
BY STDNT_ID;  
RETAIN TST_CNT TOT_SCOR;  
IF FIRST.STDNT_ID THEN DO;  
  TST_CNT = 0;  
  TOT_SCOR = 0;  
END;  
TST_CNT = TST_CNT + 1;  
TOT_SCOR = TOT_SCOR + TST_SCOR;  
IF LAST.STDNT_ID THEN DO;  
  AVG_SCOR = TOT_SCOR / TST_CNT;  
  OUTPUT;  
END;  
KEEP STDNT_ID AVG_SCOR;  
%MEND;  
XAVGSCORE(CLASS = ENGLISH)  
XAVGSCORE(CLASS = HISTORY)  

And then repeating just that one line for each class in the school. What happens in this case is that the macro processor is told that the value of an amper variable named CLASS is going to be given to it when the macro is called, and the macro processor must make the substitution.

While we have seen the power substitution gives us, there would still be limitations if that was all we could do. Fortunately, the SAS Macro language is a complete language, with variable manipulation capabilities and decision making capabilities. In other words, the SAS Macro language has capabilities that let you manipulate the values of amper variables, such as finding the length of a variable, or producing just a substring of a variable.

The language also has the capability of making decisions and determining "program flow", such as %DO loops to repeat a piece of macro code or to group macro code within an %IF-%THEN/%ELSE group. Or the ability to execute one piece of code under one condition and a second piece of code under a second condition. For example, in our evolving macro, we could test that the value passed into the macro for &CLASS is actually a valid class. We would then run the datastep if it is a valid class, and print an error message to the log if it is not.

The SAS Macro language is an extremely powerful language. The important thing to remember is that it works by substitution. The purpose of a SAS Macro is to generate SAS code that is substituted into the SAS code stream where the macro call occurs. This is explained fairly well, along with a complete discussion of the SAS Macro language and the way it works, in the "SAS Guide to Macro Processing" available from the SAS Institute.