ABSTRACT.

Learning to use macros can be an intimidating experience when one is faced with either very technical explanations or trying to decipher code written using the more advanced features of the SAS macro facility. This paper will present simple macro applications that have been used in designing standard programs. A focus will be on appropriate uses, as well as providing the inexperienced macro user with examples of SAS macro code and concepts that are easily understood.

INTRODUCTION.

When beginning to use the SAS macro facility, it is often most helpful to see how others have utilized it in specific applications that are familiar. Unfortunately, in many cases, these examples are written using more advanced features of SAS macros. Trying to read and understand another programmer's code in itself can be a trial because of differences in programming style. Additionally, when the macros are defined and called in separate sections of a program, within other macros, or in different programs altogether, it is a very intimidating environment in which to be learning a new skill.

In the Bioanalysis Department at Syntex Research, a group of individuals of varying levels of SAS expertise, undertook the project of writing a library (set) of standardized SAS programs. The objective of this project was to provide a system of software that facilitated the rapid analysis and reporting of clinical trial data for obtaining governmental approval to market an investigational new drug. Since the software needed the flexibility to analyze a variety of clinical trials, all of the programs utilized various aspects of the macro facility.

The examples presented here are derived from our experiences in developing this system and illustrate the following uses of macros:

1. String and variable substitution, accessing system variables, conditional execution of code for communication across steps, and using macros for repetitive generation of blocks of code.
2. Features of the SAS macro facility utilized in these examples include %LET, %IF, %DO, positional parameters, system variables, macro variables and macro statements.

The intention of this presentation is to provide suggestions and examples for beginning users of the SAS macro facility. We've found the techniques presented here to be both useful and a good way to start people programming with macros. Explanations of how the macro facility processes the code and the more technical explanations are left to the SAS manual. It is hoped that by following through the simple macro applications in this paper, many of these ideas can be adapted to other users needs and programs.

EXAMPLE 1: Variable/String Substitution.

One of the simplest forms of macro usage is the %LET statement in combination with macro variables. The programmer incorporates macro variables (also called symbolic variables) into places in the program that are likely to change each time the program is used. These macro variables have the value of the character string set in the %LET statement that is directly substituted into SAS code when called by prefixing an ampersand (&) to the variable name. For example, %LET STUDY=1014 defines a macro variable named STUDY with the value of 1014. When it is referenced as &STUDY in the program, the value 1014 will be substituted.

Some examples of usage are:

- Allows specific clinical study number to be input to TITLE statement. (Note: Double quotes (" ) are necessary in order for macro variable to resolve within quotes required for titles in SAS version 5. Also option DQUOTE should be set if not already a default for your system.)

```sas
%LET STUDY=1014;

< SAS code >
TITLE "CLINICAL STUDY: &STUDY";
```

Will resolve to:

"TITLE CLINICAL STUDY: 1014;"

- Inputs desired variables into a KEEP statement.

```sas
%LET VARS = INVEST PT VISIT VISDATE;

< SAS code >
KEEP &VARS;
```

Will resolve to:

KEEP INVEST PT VISIT VISDATE;
User specifies follow-up form date and converts the date from a character variable to a SAS date variable. If the date is already in a SAS date format, the %LET statement can be changed to " %LET FUDATE=FDAT10; "

%LET FUDATE=INPUT(FDAT10, YMDDMDD6.);

Will resolve to:

FORMDATE=INPUT(FDAT10, YMDDMDD6.);

Using macros for variable or string substitution offers the programmer an easy way to provide for individual 'tailoring' of a program when input to that program varies. In very long or complex programs it also allows for user changes to be placed at the beginning of a program for later substitution in the appropriate step when called. The advantage of this is that the user can make necessary changes in one place without having to read through (and understand) the entire program. Also, the macro variables serve as prompts for the user to provide the information required by the program.

Macro variables are especially effective when programs provide similar output reports but are derived from different variables. For example, in a clinical environment the format for tables reporting daytime and nighttime pain may differ only in the variables input to the program and the titles that are to be output to the table. Instead of writing and maintaining two separate programs, these differences can be accommodated in macro variables to be set by the user to indicate the desired table.

EXAMPLE II: Conditionally Execute or Select Code.

Another simple usage of macros is in directing the flow of a program. Based on the value of a macro variable set by the user at the start of a program, the program will follow different paths for building the code to execute. For example, a program may be designed to create optional tables or listings depending on user needs as specified by macro variables. The program then uses a conditional %IF-%THEN %DO-%ELSE %DO routine to provide the appropriate output. The macro statements %IF and %DO work in much the same way as the regular SAS statements IF and DO, but in addition they can be used outside of data steps to function across steps in the program. In the following example, the listings desired by the user are chosen by setting the values of the input macro variables to 1. When the program reaches the point when it is ready to output these listings, it then selects the appropriate PROC PRINT steps according to the conditions being met in the %IF statements.

*************************************************
* SPECIFY WHICH PRINTS YOU WANT PRODUCED:
** 1 - MEANS PRODUCE THE PROC PRINT
** 0 - MEANS DO NOT PRODUCE THE PROC PRINT
*

%LET PRI_ALL=1;
%LET PRI_EFF=1;
%LET PRI_DEM=0;

%MACRO LISTINGS;
%IF &PRI_ALL=1 "THEN %DO;
 PROC PRINT DATA=MASTER;
 TITLE 'DATASET #MASTER - ALL VARIABLES';
 %END;
%IF &PRI_EFF=1 "THEN %DO;
 PROC PRINT DATA=MASTER;
 VAR INV PT VISIT HRATE SBP;
 TITLE 'DATASET #MASTER - EFFICACY VARIABLES';
 %END;
%IF &PRI_DEM=1 "THEN %DO;
 PROC PRINT DATA=DEMOGR;
 TITLE 'DATASET #DEMOGR - DEMOGRAPHY DATA';
 %END;
%MEND LISTINGS;

The %IF and %DO statements provided by the macro facility can be used to conditionally build code to create datasets or run procedures. It gives SAS the ability to communicate across different program steps as well as within them.

EXAMPLE III - Repetitive Generation of Code.

Using macros to generate repetitive SAS code helps to make programs more concise and generally easier to read. In this example the formula to convert a variable from a 24 hour clock time to a SAS time variable, using the HMS (hours, minutes, seconds) SAS function, is placed in a macro %TIME. The variable [H] is a positional parameter referenced in the macro as $H whose value may vary based upon the value given when the macro is called in the program.

510
Without using macros the code looks like:

\[
\text{TIME3} = \text{HMS} \left( \frac{\text{INT} \left( \text{TESTM3} \right)}{100}, \right.
\left. \text{INT} \left( \text{TESTM4} \right) - 100 \times \text{INT} \left( \frac{\text{TESTM3}}{100} \right), \right.
\left. \text{INT} \left( \frac{\text{TESTM4}}{100} \right) \times 100 \right); \\
\text{TIME4} = \text{HMS} \left( \frac{\text{INT} \left( \text{TESTM4} \right)}{100}, \right.
\left. \text{INT} \left( \text{TESTM5} \right) - 100 \times \text{INT} \left( \frac{\text{TESTM4}}{100} \right), \right.
\left. \text{INT} \left( \frac{\text{TESTM5}}{100} \right) \times 100 \right); \\
\text{TIME5} = \text{HMS} \left( \frac{\text{INT} \left( \text{TESTM5} \right)}{100}, \right.
\left. \text{INT} \left( \text{TESTM6} \right) - 100 \times \text{INT} \left( \frac{\text{TESTM5}}{100} \right), \right.
\left. \text{INT} \left( \frac{\text{TESTM6}}{100} \right) \times 100 \right); \\
\text{TIME6} = \text{HMS} \left( \frac{\text{INT} \left( \text{TESTM6} \right)}{100}, \right.
\left. \text{INT} \left( \text{TESTM7} \right) - 100 \times \text{INT} \left( \frac{\text{TESTM6}}{100} \right), \right.
\left. \text{INT} \left( \frac{\text{TESTM7}}{100} \right) \times 100 \right); \\
\text{TIME7} = \text{HMS} \left( \frac{\text{INT} \left( \text{TESTM7} \right)}{100}, \right.
\left. \text{INT} \left( \text{TESTM7} \right) - 100 \times \text{INT} \left( \frac{\text{TESTM7}}{100} \right), \right.
\left. \text{INT} \left( \frac{\text{TESTM7}}{100} \right) \times 100 \right);
\]

After learning to use macros the code can be written:

```sas
%MACRO CTIME (N);
TIME&N = HMS (INT(TESTM&N/100),
INT(TESTM&N)-100*INT(TESTM&N/100),
(TESTM&N-INT(TESTM&N» *100);
%MEND CTIME;

%CTIME (3)
%CTIME (4)
%CTIME (5)
%CTIME (6)
%CTIME (7)
```

The SAS code generated from the resolution of this macro will be identical to the above example of code without macros.

This use of macros is especially helpful in simplifying the maintenance of code. For instance, if the formula for the conversion changes, the change would only affect the 3 lines in the macro thereby reducing the chances for error as well as making it much easier to update.

EXAMPLE IV - Repeat Blocks of Code With Variable Substitution.

This next example again utilizes the macro facility's ability to generate repetitive SAS code for execution using variable substitution. The purpose of this section of code is to create and print out separate datasets for discrepancies in the data. The macro PRINTPRB consists of a proc print step and defines 3 positional parameters to be defined in the macro call statement. The values given to these three macro variables will determine the dataset that is to be printed (CK&N) and up to two titles (TITLE and TITLE2). The datasets (CK1-CK3) are then defined, keeping only those variables that are later needed for printing. A previously created dataset (TM) is set, and based on the criteria met in the conditional IF statements the selected data is output to the appropriate dataset. The macro PRINTPRB is called and the values of &N, &TITLE, and &TITLE2 are substituted into the PROC PRINT code.

```sas
%MACRO PRINTPRB(N,TITLE,TITLE2);
PROC PRINT DATA=CK&N;
TITLE "&TITLE";
TITLE2 "&TITLE2";
%MEND PRINTPRB;

DATA CK1 (KEEP=INV PT VISIT TREAT DATE TIMEDIFF TIME1 TIME2)
CK2 (KEEP=INV PT VISIT TREAT DATE TIME1)
CK3 (KEEP=INV PT VISIT TREAT DATE);

SET TM;

IF (ABS(TIMEDIFF) GE 0.1) THEN OUTPUT CK1;
IF TIME1 = . THEN OUTPUT CK2;
IF DATE = . THEN OUTPUT CK3;

%PRINTPRB(1,DISCREPANCY IN TIMES > 0.1 MIN, CHECK FORM FOR ERROR)
%PRINTPRB(2,MISSING VALUE OF EXERCISE DURATION)
%PRINTPRB(3,MISSING DATE OF TEST)
```

The macro PRINTPRB resolves to produce the following SAS code:

```sas
PROC PRINT DATA=CK1;
TITLE "DISCREPANCY IN TIMES > 0.1 MIN, CHECK FORM FOR ERROR";
PROC PRINT DATA=CK2;
TITLE "MISSING VALUE OF EXERCISE DURATION";
TITLE2 "";
PROC PRINT DATA=CK3;
TITLE "";
TITLE2 "MISSING DATE OF TEST";
```

EXAMPLE V - Accessing System Variables.

SAS system variables (also referred to as automatic macro variables) all begin with the letters SYS and can be accessed anywhere in a program by prefixing an ampersand (&) to the variable name. A complete list of those variables available can be found in the SAS manual. This example uses SYSDATE which when referenced will give the date that the job started execution.

```sas
TITLE "JOB LAST RUN: &SYSDATE";
```

Will resolve to:

```sas
TITLE "JOB LAST RUN: 02FEB86";
```
System variables provide access to information which is useful for logging or documenting job execution runs within the program.

CONCLUSION.

A good way to begin using macros is by incorporating the following basic features of the SAS macro facility: %LET to define the macro variables and assign a value to them at the start of a program; communicating across steps in the program with conditional %IF statements and constructing blocks of code with %DO loops; using macros to repeat blocks of code while substituting values for the positional parameters defined; and calling system macro variables to provide documentation of system or job execution information.

Macros do not always work predictably and the options MPRINT and SYMBOLGEN are invaluable for debugging. They display the SAS code as the SAS compiler sees it, showing the statements produced by the macros and macro variable substitution. For a more thorough understanding of macros and how they resolve, becoming familiar with the role of the SAS macro processor and SAS processing in general is encouraged.

Although good SAS programming does not require using macros, macro code can be used to provide easier maintenance, flexibility and 'user friendliness.'

ACKNOWLEDGMENTS.

I would like to acknowledge David Tappe, Dr. Martin Rosenberg and Virginia Ozer of Syntex Research for their input and encouragement.

*SAS is the registered trademark of SAS Institute Inc., Cary, NC, USA.

For further information or comments, please contact the author at: Syntex Research Mailstop A3-434 3601 Hillview Ave. Palo Alto, CA 94303