SAS macro language gives the opportunity to insert programming steps inside a SAS code.

The first advantage is to simplify large programs. Indeed, you can create functions with parameters, which can be easily invoked several times.

It is therefore possible to define a model and submit it to statistical testing without having to use predefined functions. All the usual programming elements: if then else statements, loops for I=1 to N do, and other similar operators can be used to this effect.

One of the criticisms you could however address is it creates lines of code that are often understood only by the user itself and are therefore not easily adaptable.

It is important to have good understanding of the classical SAS language data steps and procedures to start with the macro language.

The different elements of the macro language are: Macro variables, macro program statements, macro functions.

The same structure of examples will be used throughout this document.

There are 26 data sets named year, year75, year76...year99, which all contain variables rainfall, temperature and pressure for the years 1975 to 1999. Basic procedures will be run on these data sets to illustrate the concepts introduced.
Macro Variables

A macro variable is a good example of a tool provided by the macro language that greatly simplifies a program. It assigns a value (either string or integer) to a variable that can be thereafter invoked several times.

Define a Macro Variable

To define a macro variable use the %let statement.

For example to assign the value 24 to the variable named sto do:

\[
\text{%let sto}=24;
\]

The variable sto will take this value every time it is invoked until another statement changes it. This code can be placed anywhere in the program except within data lines.

Macro variables that contain entire sections of a SAS program can also be created:

\[
\text{%let sto2=\%str(proc means data=year;}
\text{\hspace{2cm}var rainfall;}
\text{\hspace{2cm}run;}
\text{\hspace{2cm}});
\]

When the macro variable is invoked, the processor will insert the lines:

\[
\text{proc means data=year;}
\text{\hspace{2cm}var rainfall;}
\text{\hspace{2cm}run;}
\]

We used to define this macro variable, the macro function %str (see later section).

Invoke a Macro Variable

To invoke a macro variable place a & before its name. The macro processor resolves this reference to the variable by replacing &sto by the value of sto.

For example:

\[
\text{%let sto=year;}
\text{\hspace{2cm}proc means data=&sto;}
\]
runs a proc means on the data set year. The processor reads:

```
proc means data=year;
```

A same macro variable can be invoked several times.
You can also invoke a macro variable inside a statement where you are defining another macro:

```sas
%let sto=year;
%let nest=%str(proc means data=&sto;
    var rainfall;
  );
```

If you invoke the macro variable `nest` (by a `&nest` statement), the processor will write:

```
proc means data=year;
    var rainfall;
```

### A Simple Test

It is convenient to test if the macro variables have been correctly defined. Use the `%put` statement that writes text to the SAS log.

```sas
%let sto=you;
%let lol=me;
%put &sto ,,, &lol;
```

This writes in the SAS log

```
you ,,, me
```

### Macro Program Statements

A macro program is a text identified by a name. These lines of codes can be invoked several times by using this name. So, in its most simple form, a macro program is equivalent to a macro variable. However a macro program can include more complex functionalities, such as if then loops.

### Define a Macro Program

This is an example of a simple macro that could also be defined by a macro variable:
%macro example;

    proc means data=year;
    var rainfall;

%mend example;

The definition of a macro must begin by a %macro statement followed by the name of the macro. Here, our macro has been named example. The statement must end by a %mend statement. When it is invoked, the macro processor will run the lines of text between the %macro and %mend statements.

We see that the macro defined previously is equivalent to the macro variable exemple2:

%let example2=%str(proc means data=year;
    var rainfall;
    );

**Invoke a Macro Program**

To invoke a macro program, place a % sign before its name. The macro example will be invoked in the following way:

%example;

This line of text can be included at any time in the program. For example:

    Proc plot data=year;
    plot rainfall*temperature;
    %example;
    Proc print data=year;

The processor will execute the following lines:

    Proc plot data=year;
    plot rainfall*temperature;
    Proc means data=year;
Macro variables can be invoked inside a macro program. In fact, a macro program can even be invoked inside another macro program:

```sas
%let var1 = rainfall;
%let var2 = temperature;

%macro design;
   proc plot data = result;
      plot &var1 * &var2;
   %mend design;

%macro compile;
   data result;
      set year;
      keep &var1 &var2;
   %design;
   %mend compile;
```

If elsewhere in the program, compile is invoked by the statement `%compile`, the processor will replace these lines by:

```sas
data result;
   set year;
   keep rainfall temperature;
proc plot data = result;
   plot rainfall*temperature;
```

### Macro with Parameters

There is an easier solution than invoking macro variables inside the definition of a macro program: use parameters in the definition of your macro.
For example, to redefine the macro design with parameters, type the following lines:

```sas
%macro design1(para1,para2);
    proc plot data=year;
        plot &para1*&para2;
    %mend design1;
```

To invoke it, we will type: `%design1(rainfall,temperature);`

This gives the parameter para1 the value rainfall and para2 the value temperature (the processor assigns the values to the parameters in the order entered).

The result of this line of code will be equivalent to the `%design` used previously, without having to define the macro variables var1 and var2.

**Programming Steps Inside a Macro Program**

As indicated earlier on, SAS macro language offers the possibility of inserting steps of programming inside data analysis. I will give a few examples of the tools that can be used.

**%do Statement**

This functionality is best introduced by an example: to obtain the mean of the variable rainfall for all the data sets year75…year99, type:

```sas
%macro year;

    %do i=75 %to 99;
        proc means data=year&i;
            var rainfall;
    %end;

%mend year;
```
A %do statement will always be terminated by a %end statement. The counter is then referenced by a &i inside the macro.

**%if %then %do Statement**

This tool is used to insert conditional statements in a macro program. To illustrate this new tool, an extension of the previous example is used.

Suppose that it is necessary to obtain the average rainfall for every year, but that graphs are needed only for the years after 1990, then type the following macro:

```
%macro analyze;
    %do i=75 %to 99;
        proc means data=year&i;
        var rainfall;
        %if i>=90 %then %do;
            proc plot data= year&i;
            plot rainfall*temperature;
        %end;
    %end;
%mend analyze;
```

We can also mention that %do %while and %do %until statements exist.

**Loops on Series of Macro Variables**

Suppose a series of macro variables m1, m2, m3…m25 has already been defined. If they represent the rainfall predictions produced by a model, it is interesting to compute the difference between the real value and the prediction. This can be done using a loop on these macro variables:

```
%do j=1 %to 25;
    data compare&j;
```
set year\&j;

dif=rainfall-&m\&j;

\%end;

You can see that this can be done by placing two & before the name of the macro variable. The processor resolves it in the following way:
If for example j=3, it replaces first &m&j by &m3, and then scans it again to replace &m3 by the value it has been previously assigned.

Macro Functions

Macro functions process macro expressions, called arguments, to produce a new expression. We already mentioned in a previous section, the macro function %str.

Macro functions can be divided into three categories: character functions, evaluation functions and quoting functions.

Macro Character Functions

This type of macro function provides information about the string it takes as argument (see following example: the %length function). For a more extensive list and more detailed information, see the ‘SAS Guide to Macro Processing’.

The %length function returns the length of the string it takes as an argument.
In the following example the objective is to determine if the length of a variable name is smaller or longer than 8 characters (some softwares don’t accept variable names longer than 8, so it can be necessary to change the names before exporting the data set) and replace the variable by the same one but with a new name.

\%macro export (name,newname);

  %if %length(&name)>8 %then %do;
    data year;
    set year;
    &newname=&name;
    drop &name;
  %end;

%end;

%mend limit;

Later in this section the character macro function %scan (which is also a character function) will be used.
Macro Evaluation Function

The evaluation function is the %eval function. It evaluates arithmetic and logical expressions in the macro language. It only performs integer arithmetic.

Example of an arithmetic expression resolved:

In the following example, it is supposed that a certain procedure has produced an integer stored in the macro variable base (it can be as in the next example, an estimation of the number of years required for a specific analysis). If 3 is added to this integer, we could obtain the year up to which proc means have to be run so as to remove the possible estimation errors in the parameter base.

\%
macro add(base);

   %let result=%eval(&base+3);
   %do i=75 %to &result;
      proc means data=year&i;
      var rainfall;
   %end;

%mend add;

We see that once the macro expression is evaluated, the result can be stored in another macro and can be used immediately.

Logical expression can also be evaluated:

\%
macro biggest(a,b);

   %let logic=%eval(&a>&b);
%mend biggest;

Macro Quoting Functions

These functions enable us to be sure that the macro processor will treat the argument as text and will not perform any kind of evaluation or calculation over it.
In the first paragraph of this presentation, the macro function `%str` was used. This is a good example of a quoting macro function. The argument of this macro function can be an entire section of a SAS program. This was the example presented earlier.

**An Interesting Application: The Use of Lists in a Macro Program**

A list is defined as a string where each element is separated by a blank. The macro functions defined for string manipulation are used to work on these lists.

For example a list of numbers can be defined as follows:

```
%let numbers=one two three four five six seven eight nine;
```

The elements of this list are then selected by using the `%scan` function.

The syntax of this function is `%scan(argument,n)`. It selects in the string that is given by argument, the \(n\)th element (the different elements being separated by blanks). The processor replaces immediately the expression by its result.

So, we can have the following macro:

Suppose the data sets are not named year81 year82….but yeareightyone, yeareightytwo;

```
%macro auto(list);

%do j=1 %to 9;
    proc means data=yeareighty%scan(&list,&j);
    var rainfall;
%end;

%mend auto;
```

The command `%auto(&numbers)` will apply this macro to the list numbers and will therefore run proc means on the data sets yeareightyone, yeareightytwo…

**To Retrieve a Value from a Data Set and Place It in a Macro Variable**

Macro language is often used to generate automatic operations based on existing data set. It is therefore often necessary to select certain values in the data set to work on them.

This is done through a call `symput` statement: `call symput(macrovariable name,value)`. It places in the macro variable given by the first argument, the value given by the second argument.
To select the first observation of the variable rainfall and place it in the macro variable result, use the following statement.

```sas
%macro select;

data inter;
set year75(firstobs=1 obs=1);
keep rainfall;

data inter;
set inter;
call symput(result,rainfall);
%mend select;
```

The first data step, selects the first observation of all variable and then keeps only the variable rainfall. After this step, the data set inter only has one entry. The second data step is only used to store this observation in the macro variable. Because only one observation is left, the value is given by the variable itself. The selection of the desired observation is a required preliminary step. This is done using the statement (firstobs= obs= ), where firstobs indicates the first observation you want to select and obs the total number of observations you want.

This last example is a more complex illustration of the principles described before. Suppose that the data sets year1…year25 also contain the variables named var1…var10. To store the first observation of all these ten data variables in all these data sets, the following procedure is used:

```sas
%macro read;

%do j=75 %to 99;

%do k=1 %to 10;

data inter;
set year&j(firstobs=1 obs=1);
keep var&k;

data inter;
set inter;
```

SAS Macro Language – 11
call symput("y&l" | "v&k",var&k);

%end;

%end;

%mend read;

First, it is important to remember that the operator || sticks two strings together to produce a single one.

In this example, the value of the first observation of the variable var4 in the data set year81 is named y81v4.

In exactly the same way, it is possible to select the elements of a matrix and store them in macro variables named a_{ij}.

For More Information and Assistance

SAS Documentation and Books

Please see the document “Resources for Learning SAS” on SSDS website.

SSDS Software Services at Stanford

Software Services provides technical support for SAS users at Stanford. Users can ask questions or make appointments with the consultants via our website. For more information or to contact us, see the web at:

http://ssds.stanford.edu/

Note: This document is based on SAS 9.

Copyright © 2001, 2005 by The Board of Trustees of the Leland Stanford Junior University. Permission granted to copy for non-commercial purposes, provided we receive acknowledgment and a copy of the document in which our material appears. No right is granted to quote from or use any material in this document for purposes of promoting any product or service.

Social Science Data and Software
Document revised: 2/1/2008