Paper BB-04

Using Recursion for More Convenient Macros

Nate Derby, Stakana Analytics, Seattle, WA

ABSTRACT

There are times when a macro needs to alternatively be applied to either one value or a list of values. In this case, adding a recursive definition to a macro can make it easily accommodate both situations. This can be particularly useful for testing and investigative purposes, as explained in this paper.

KEYWORDS: SAS, macro, code, recursion.

This paper is an excerpt from Derby (2010), which may be updated and can be downloaded for more information.

INTRODUCTION: WHY USE RECURSION?

For flexibility, efficiency and convenience, it's useful to define a macro such that it can be used for both a general and for a specific case. As an example, let's suppose we have a macro for making airline demand forecasts, `%makeForecasts`, which accesses global macro variables `&orig` (origin) and `&dest` (destination), with values of YYC and YVR, respectively. That is, we are looking at flights from airport YYC (Calgary) to YVR (Vancouver, BC). Now suppose our `%makeForecasts` macro uses a parameter for a certain flight number:

```
%makeForecasts( fnumber=1542 )
```

In this case, forecasts would be generated for flight number 1542 only. If we wanted forecasts to be made for all flights from YYC to YVR, we would have to first find all flight numbers for flights on that leg, and then loop through them:

```
%makeForecasts( fnumber=1542 )
%makeForecasts( fnumber=1543 )
%makeForecasts( fnumber=1544 )
```

However, this is tedious for two reasons: We have to list them out individually, and we have to find the flight numbers pertaining to our given origin (`&orig`) and destination (`&dest`) airports. It would be much easier if we could just have our code automate all that by simply using the macro without parameters:

```
%makeForecasts
```

Then we could just change the value of macro variables `&orig` and `&dest` to implement this change in `%makeForecasts`. However, it would be useful to *also* be able to run it individually for a specific flight number, as done for flight 1542 above. This is useful for decreasing run time for testing and drill-down purposes:

- When writing and testing the code, we can test it for one flight number before testing it for all of them at once.
- For drill-down capabilities, we can use this when looking just at a specific flight (e.g., flight 1542).

In practice, it's most useful for a combination of the above two reasons. For instance, if we are specifically interested in flight 1542, we can drill down by first looking at the results for that specific flight number, then tinker with the macro for that flight number (e.g., looking at some of the forecasting methods in detail).

USING RECURSION

This dual functionality can be accomplished by *defining a macro function recursively*. That is, we include a reference to itself within its definition. We can do this via three steps:

```
%MACRO makeForecasts( fnumber=all ); ❶

  %LOCAL i n fnumbers; ❷

  %IF &fnumber = all %THEN %DO; ❸

    %getFlightNumbers; ❹

    %LET i = 1;
    %DO %WHILE( %LENGTH( %SCAN( &fnumbers, &i ) ) > 0 ); ❺
     %LOCAL fnumber&i;
     %LET fnumber&i = %SCAN( &fnumbers, &i );
     %LET i = %EVAL( &i + 1 );
     %END;
    %LET n = %EVAL( &i - 1 );

    %DO i=1 %TO &n; ❻
     %makeForecasts( fnumber=&&fnumber&i );
     %END;

    %GOTO theend; ❼

    %END;

  [Code for making forecasts, made in step 1]

  %PUT fnumber=&fnumber; ❽

  %theend: ❾

%MEND makeForecasts;
```

Figure 1: Adding a recursive definition to the macro `%makeForecasts`.

1. First, we define the macro for the parameter value in question:

```
%MACRO makeForecasts( fnumber );

  [Code for making forecasts]

%MEND makeForecasts;
```

2. Then we create a small, auxiliary macro to determine our list of parameter values (in this case, flight numbers). There are many ways to do this – one is with `PROC SQL`:

```
%MACRO getFlightNumbers;

  PROC SQL NOPRINT;
    SELECT DISTINCT flightnumber INTO :fnumbers SEPARATED BY ' '
      FROM datasource
      WHERE orig="&orig" AND dest="&dest"
      ORDER BY by flightnumber;
  QUIT;

%MEND getFlightNumbers;
```

3. Lastly, we change our macro definition as shown in Figure 1. Note the following:

   ❶ Adding `fnumber=all` gives us a default parameter value, `all`, that would never happen as a natural value for a flight number. As such, this can be thought of as a value which tells us to use the recursive definition.

   ❷ Following best practices, we want to keep all internal macro variables local.[1] The macro variable `&fnumbers` is defined within the `%getFlightNumbers` macro at ❹ and accessed at ❺.

   ❸ Here is where we include the recursive definition. Code within this `%IF ... %DO` block is skipped whenever a valid (i.e., numeric) value of `&fnumber` is given.

   ❹ This is the macro we defined in step 2, for finding the flight numbers.

   ❺ Here we define one macro variable, `&&fnumber&i`, for each possible flight number.[2]

[1] A *local* macro variable is only defined within the macro, whereas a *global* one is defined outside of the macro.

[2] The double ampersand just tells us to resolve that after the single ampersand variables are resolved. Thus, looping through the vales of the macro variable `&i`, we are creating macro variables `&fnumber1`, `&fnumber2`, etc. For more information, see Carpenter (2004) or Burlew (2007).

❻ Here we loop through all possible flight numbers that we defined in our previous step, calling itself at each step.

❼ After we are done with the loop, we need to skip to the end of this macro, since we are essentially done.

❽ For testing purposes, we use a `%PUT` statement like this one to print our parameter values onto the log.

❾ This is the end of our macro definition.

Now running this macro without a parameter shows us in the log that this runs correctly:

Macro call		Log output

`%makeForecasts`	⇒	`fnumber=1542`
		`fnumber=1543`
		`fnumber=1544`

Running this macro with a parameter gives us output for that flight number only:

Macro call		Log output
`%makeForecasts( fnumber=1542 )`	⇒	`fnumber=1542`

RECURSION ON MULTIPLE PARAMETERS

We are not limited to one parameter value for this technique; we can define a recursion with any number of parameters. Suppose that `%makeForecasts` loops through three different forecasting methods: `AddPick` (Additive Pickup), `ExSm` (Exponential Smoothing) and `ARIMA` (ARIMA). We want to design it so that it can loop through any one of them individually, as well as looping through the flight numbers individually as before. Figure 2 shows how we can modify our code in Figure 1 to also loop through the methods. This way, the statement with no parameters listed gives us all flight numbers and methods:

Macro call		Log output

`%makeForecasts`	⇒	`fnumber=1542, method=AddPick`
		`fnumber=1542, method=ExSm`
		`fnumber=1542, method=ARIMA`
		`fnumber=1543, method=AddPick`
		`fnumber=1543, method=ExSm`
		`fnumber=1543, method=ARIMA`
		`fnumber=1544, method=AddPick`
		`fnumber=1544, method=ExSm`
		`fnumber=1544, method=ARIMA`

whereby running it with one parameter gives us output for that parameter value only:

Macro call		Log output

`%makeForecasts( method=ARIMA )`	⇒	`fnumber=1542, method=ARIMA`
		`fnumber=1543, method=ARIMA`
		`fnumber=1544, method=ARIMA`

`%makeForecasts( fnumber=1542 )`	⇒	`fnumber=1542, method=AddPick`
		`fnumber=1542, method=ExSm`
		`fnumber=1542, method=ARIMA`

and running it with both parameters gives us output for those two parameters only:

Macro call		Log output
`%makeForecasts( fnumber=1542, method=ARIMA )`	⇒	`fnumber=1542, method=ARIMA`

```
%MACRO makeForecasts( fnumber=all, methods=all );

  %LOCAL i n fnumbers methods; ❶

  %IF &fnumber = all %THEN %DO; ❷

    %getFlightNumbers;

    %LET i = 1;
    %DO %WHILE( %LENGTH( %SCAN( &fnumbers, &i ) ) > 0 );
     %LOCAL fnumber&i;
     %LET fnumber&i = %SCAN( &fnumbers, &i );
     %LET i = %EVAL( &i + 1 );
     %END;
    %LET n = %EVAL( &i - 1 );

    %DO i=1 %TO &n;
     %makeForecasts( fnumber=&&fnumber&i, method=&method ); ❸
     %END;

    %GOTO theend;

    %END;

  %IF method = all %THEN %DO;

    %LET methods = AddPick ExSm ARIMA; ❹

    %LET i = 1;
    %DO %WHILE( %LENGTH( %SCAN( &methods, &i ) ) > 0 );
     %LOCAL method&i;
     %LET method&i = %SCAN( &methods, &i );
     %LET i = %EVAL( &i + 1 );
     %END;
    %LET n = %EVAL( &i - 1 );

    %DO i=1 %TO &n;
     %makeForecasts( fnumber=&fnumber, method=&&method&i ); ❺
     %END;

    %GOTO theend;

    %END;


  [Code for making forecasts]


  %PUT fnumber=&fnumber, method=&method;

  %theend:

%MEND makeForecasts;
```

Figure 2: Adding a two-parameter recursive definition to the macro `%makeForecasts`.

A few things to note about the code in Figure 2:

- ❶ We need to add a new local macro variable for our new list of parameter values (`&methods`).
- ❷ It doesn't matter what order we put the parameters. Here the flight numbers are the first level and the methods are the second, giving us the order shown in the log output shown on the preceding page.
- ❸ When calling the macro, keep track of which parameters are being looped. Here, we are looping through the flight numbers (`&&fnumber&i`) and keeping the method constant (`&method`). This is very different from the call at ❺.
- ❹ For simplicity, we are listing the forecasting methods individually in a macro list (which couldn't be done with the flight numbers because that was dependent on the data). If this list is accessed from multiple macros, it's best for code maintenance to put this into a separate macro, which could be called `%getForecastMethods`.
- ❺ As opposed to ❸, we are now looping through the methods (`&&method&i`) and keeping the flight number constant (`&fnumber`).

CONCLUSIONS

Using a recursive definition of a macro can make the macro more flexible, efficient and convenient for testing and investigative purposes. Recursion can be applied to one or many parameters. Furthermore, it can work especially well within a larger framework for using macros, as explained in Derby (2010).

REFERENCES

Burlew, M. (2007), *SAS Macro Programming Made Easy*, second edn, SAS Institute, Inc., Cary, NC.

Carpenter, A. (2004), *Carpenter's Complete Guide to the SAS Macro Language*, second edn, SAS Institute, Inc., Cary, NC.

Derby, N. (2010), Suggestions for organizing SAS code and project files.
`http://www.nderby.org/docs/DerbyN-SASOrg-cur.pdf`

ACKNOWLEDGMENTS

I thank my present and past employers and clients for giving me such a high degree of flexibility in organizing my projects. I very much thank SAS technical support for helping me learn the basics of macro programming. Lastly, and most importantly, I thank Charles for his patience and support.

CONTACT INFORMATION

Comments and questions are valued and encouraged. Contact the author:

Nate Derby
Stakana Analytics
815 First Ave., Suite 287
Seattle, WA 98104-1404
`nderby@stakana.com`
`http://nderby.org`
`http://stakana.com`