Abstract:
This paper will discuss certain things to keep in mind while designing SAS programs using SAS Macros.

The SAS Macro facility is in essence a pre-processor which allows programmers a way to neatly bundle similar lines of code together and execute them by invoking a macro call. Some programs do not require the use of the macro facility, and therefore, should not use them.

The use of SAS Macros require more advanced skills particularly in the debugging phase. This is true because the line number reference you receive back in the SAS Log refers only to the line where the macro was invoked. Meaning, the error message does not have a corresponding line number.

If you get a syntax error back and you cannot find where it is coming from, submit the SAS code outside a macro definition.

Macros can span across multiple data steps and proc steps. Macros can encompass just a few lines of code or thousands of lines of code. Macros can contain all macro language references or mostly BASE SAS language. These issues are not really important. What is important is that the answer to one of the following questions is YES.

1) Does the macro consolidate similar lines of code together (thus minimizing the # of lines of code what are maintained)?

Let say you have a program which has five different summaries and prints. This can be consolidated as follows:

```sas
%macro summ(dsn=phys, vars=amtpaid charges, class=month, title=, stat=sum)
   proc summary data=&dsn nway;
   class &class;
   var &vars;
   output out=&dsn.out &stat.=
   proc print;
   title1 ""&title;"
   %mend summ;
```

2) Does the macro encapsulate a detailed function (which is simplified by having programmers only understand the call)?

3) Does your macros segment an information system into logical modules?

If the answer to all the above questions is no, it is more likely that the macro you have written is not designed properly.

This leads us to the next topic: How to design macros as logical modules. Logical modules create segments of a program or system which can be thought of logically without needing to know all the detail. These logical modules allow us to create "simplified building blocks" as reusable tools.

For example, we are writing a batch production system which reads several flat files; performs a variety of data manipulation and then joins and/or selects the data prior to summarizing into a variety of tables.

In the above example, the simplified building blocks would be:

* input macros one for each flat file
* a data manipulation macro (possibly one per file depending on complexity)
* a join and/or select macro
* a summary macro
* table macros

This approach allows the programmer to more readily modify and maintain this code. If someone comes to you with a modification, the programmer can quickly find where he needs to make the change because the system (even if its in excess of 10,000 lines) has logical "simplified building blocks."

Conclusions:
* Use Macros to simplify software solutions
* Use Macros to minimize the number of lines of code to maintain
* Use Macros to organize systems into "Logical Building Blocks"