This program was designed for test purposes, but it demonstrates a typical enough kind of application. The initial data step reads in a set of variables; for testing purposes, the program uses card input. Figures 2 and 3 show the input and output data sets. The body of the program takes place in Data step "TWO." The function of this program is simply to compute the percentage change between the current observation and the previous observation. The array "V" contains the 26 variables on which we wish to compute percentage changes. Array "L" holds the 26 val us of prevous observations; notice in statement 25 that the "L" array needs to be retained because otherwise it will be reset to missing each time the SET statement is executed. The "V" array at line 26 holds the computed percentage changes. The loop at statements 21 through 24 repeats the computation of the 26 percentage change variables, which are equal to 100 times the difference between the previous value and the current value over the previous value. The "L" value is set equal to the "V" value, so that on the next iteration of the data step, the value is logged.

P1 through P26 now contain the 26 percentage changes. Figure 2 shows the 26 variables: there are six observations. I confess to a lack of imagination in giving the variables these values, but they will do for test purposes. The output data set shows the percentage changes. The values for observation 1 are always missing because there are no previous values. The value for observation 2 for P1 is the difference between 1 and 26, which is 2,500%. The difference between 26 and 1, P26, is -96.2%.

This program, while useful, is not very flexible; it has a number of disadvantages. For one thing, the list of the 26 input variables appears twice in the program, in both the first data step and in the second data step. Furthermore, the number 26 appears six times in the program. It appears in statement 13, statement 20, statement 21, statement 25, statement 26 and statement 29. Even if one only knows a little about the SAS Macro language, it should be fairly obvious that one can make some improvements in this program simply by defining a couple of substitutable strings. For example, the list of variables that appears twice in the program, as shown in lines 1 thru 5 of Figure 6, can be replaced with a macro called "VARS" naming the list of variables to be processed by the program. Now, VARS appears in line 8 of the initial step and in line 23 where we define the "V" array. In this particular program, VARS is going to be exactly the same in both places, so one can type it once and put the macro in in both places, thus avoiding typing those 26 names over again. At line 6, we have a "LET" statement that creates a macro variable called "N." Its value is the character string "26." It doesn't have single quotes around it because all macro variables are character strings. But
remember that it is not the number 26, it is the EBCDIC characters "two six." The macro variable is used at the 6 places in the program at which the number "26" appeared in the first example. At lines 31 and 33 the arrays are dimensioned &N rather than 1126. We do "N equals 1 to N in line 33, and in 40 and 42 we keep and retain LI thru 26N. In the Print processor, the format describes PI thru P&'N. The macro processor simply goes through this program at the outset and every place it finds an &:N it sticks in a character string that is substituted. But there is something obscurely upsetting about this program; that is, the program ought to know there are 26 variables. Currently, it is necessary to count them up and specify their dimension in statement 6. What if you count them wrong? This paper started out by trying to illustrate less error prone code -- why doesn't the program count how many things there are in the variable list? The next version of the program (Figure 3) does precisely that. In lines 1 thru 6, the macro DIM, has been added to the previous version of the program. Comparing Figure 4 with Figure 5, it is evident that lines 7 thru 11 are still the macro VARS, unchanged from the previous example. This time, N is initialized to "N" as a global macro variable. (If N were initialized inside the macro, it would be a local variable, and would be undefined in the global context.) SAS skips a couple of lines in between lines 13 and 16 because it wrote some macro code in there that is invisible (with the Macrogen option turned off). By the time the program gets to line 16, it knows that N is equal to 26. That is, the purpose of the macro DIM is to assign to the macro variable N the character string 26. The entire function of the macro is to count the number of items in VARS and assign that value to N. 

The program is now (to all intents and purposes) pretty flexible, because all one has to do is change the first six lines and the rest of the program will work correctly no matter what variables are listed or how many of them there are. But there is something obscurely upsetting about this program; that is, the program ought to know there are 26 variables. Currently, it is necessary to count them up and specify their dimension in statement 6. What if you count them wrong? This paper started out by trying to illustrate less error prone code -- why doesn't the program count how many things there are in the variable list? The next version of the program (Figure 3) does precisely that. In lines 1 thru 6, the macro DIM, has been added to the previous version of the program. Comparing Figure 4 with Figure 5, it is evident that lines 7 thru 11 are still the macro VARS, unchanged from the previous example. This time, N is initialized to "N" as a global macro variable. (If N were initialized inside the macro, it would be a local variable, and would be undefined in the global context.) SAS skips a couple of lines in between lines 13 and 16 because it wrote some macro code in there that is invisible (with the Macrogen option turned off). By the time the program gets to line 16, it knows that N is equal to 26. That is, the purpose of the macro DIM is to assign to the macro variable N the character string 26. The entire function of the macro is to count the number of items in VARS and assign that value to N.
It works by executing the macro language code that you see at lines 2 thru 4. Line 3 shows how you increment a macro variable. This is covered in the macro language reference section of the base SAS Reference Guide. If you say "N = N+1," the value of N will be the character string "N+1." If you say "N = &N+1," the result of N will still be a character string "&N+1." It is important to emphasize than using &N+1 to assign a value to a macro variable is only the way SAS assigns a character string to that variable. So in order to add 1 to N, you need to use a macro language function. There are a number of macro language functions described in the section on macro language function in the Base SAS Reference Guide. One of these is %EVAL, which returns the numeric value of a macro language expression. So %EVAL(N+1) is going to be evaluated by the macro execution phase into whatever the current value of N is plus 1. In line 12, N is set to be equal to 0; the first time the macro loop is executed, that will be translated into 0 plus 1, or 1. Thus, 1 is assigned as the value of N. The second time it is executed, N will be equal to 1. This parenthesis will evaluate to 1 + 1, that is 2, so 2 will be assigned to N. The %EVAL function is how you add to macro variables.

Statement 3 occurs inside a macro loop, that continues until the expression in parenthesis on line 2 results in an expression that is not true. In other words, it will continue as long as that expression is true, just like any other "DO WHILE." In other words, it will continue as long as the expression is true, just like any other "DO WHILE." This loop is only going to be executed at one place in the program no matter how many input observations there are in the input data set. The loop is going to count how many things there are in the list, and then it is going to rewrite the SAS code in the rest of the program. Every place it sees &N it is going to stick in 26.

Now to make some sense out of that expression in parentheses at line 2. Starting at the right hand end, whatever is to the left of that "&N" sign has got to be some kind of character string. That character string is being tested to determine whether it is greater than a blank. If it is blank, then the test is false because blank is not greater than blank, since everything in the EDDIC collating sequence is greater than blank. If it is blank, then the test is false because blank is the lowest character string. We know from statement 3) that %EVAL(&N+1) returns a number that is greater than the current value of N by 1. Furthermore, we have seen that %VARS is our list of variables. That just leaves %SCAN, another of the SAS macro functions (documented in the same place as %EVAL). %SCAN takes two arguments, Argument 1 is a list of things and Argument 2 is a number. So %VARS is the list and N + 1 is the number; %SCAN returns the ith member of the list. It counts through the list by looking for delimiters, which are spaces or commas, you can even specify what kind of delimiters you want it to use. In the VARS macro (line 3 thru 10) each one of the items in the list is separated by a blank.
The first element that is returned by the %SCAN function from %VARS is "VA" and the last one it returns is "VZ." If $N + 1$ is equal to 27, the %SCAN function returns a blank, because there is not a 27th element in that list. That is how the loop is halted. If $N$ is 0, the first time statement #2 is executed, the first item in the list is "VA." When $N$ is 26, $N + 1$ will be the 27th item (a blank) and so it will stop. $N$ is 26 and that is how many items are on the list.

The rest of the program is identical to the one shown in Figure 4. In line 12, $N$ is initialized as 0 (not 26). At line 13, DIM is called, returning a value of 26. In data step two, &N is substituted for all of those 26s in the former program. Line 34 is the array %VARS. The next line number is number 40 (because the macro language is putting some lines in there that are invisible). At line 40 and 42, we dimension the arrays L and P with &N. The only difference between the program in Figure 6 and that shown in Figure 5 is that macro lets the program figure out how many items there are in the list. At line 5, we have a %PUT statement. The %PUT is inside the macro DIM, but is executed after the loop; its result appears following line 11. Interestingly enough, it appears in the log two lines ahead of where DIM is actually called. It always does that. That is to remind you, I suppose, that what you see is not necessarily what you get. The macro preprocessors are being executed prior to the data one step, but I think the SAS log does not get written until data one step executes it. Anyway you see that it is counted right. %VARS is defined prior to the execution of dim. (I said %DIM and then %MACRO VARS, it would not work. The macro called DIM and the macro called VARS, do not happen until line 13. VARS has to be defined when I say %DIM, but DIM does not have to be defined prior to %VARS.

Now we have a program that is pretty smart. You feed it VARS which is any arbitrary list and it will go ahead and run no matter how many items are in the list. I would argue that we have now made this program much smarter than it was before. As Figure 6 shows, however, this program can be made even smarter. The rest of the program following line 16 is exactly identical to the prior versions. Note the difference between the DIM macro in this program and that of the previous program (Figure 5). At line 13, the %DIM has been replaced by %DIM (VARS, N). The difference is that the new code is passing parameters to the macro. These parameters are themselves macro variables. The macro in Figure 3 required that the list be called VARS and that the macro variable be called N, because they were built into the macro. This new macro doesn't care what your list is called and it doesn't care what your macro variable list is called. It passes the names of the list and the names of the macro variables as parameters to the macro. It is now truly a general purpose macro; lines 1 thru 6 can now exist independently of any particular program. You have written a real function! The function takes two arguments: a list and a variable. Values of those macro variables are themselves macro statements; in the first case a macro name VARS, in the second case, a variable named N.

Each statement in Figure 6 corresponds to one in the Macro shown in Figure 5. Starting with line 5 for simplicity's sake, the number of items in &LIST is &1. &LIST evaluates to the first parameter, that is "VARS," &1 is equal to "N." (Inside a Macro two ampersands always resolves to a single one, so &1 is &N.)

The same principle applies to statement 3. &2&1 is &N. In this case, &1 is equivalent to N; you can use one ampersand or two, it works both ways (since two are always resolved to one). Using these substitution rules in statement 2 results in similar equivalences. %&list is % vars, while &1&1 (as in statement 5) is &N.

Since this is now a universal function, it can be stored in a library of similar functions. Figure 7 replaces the macro with a %INCLUDE statement. "Macros" is a file name (or dname in MVS) referencing a sequential file that contains the code that appeared in lines 1-6 of Figure 6. The library of Macro functions can be made available to all of the users at an installation, who need not be concerned with the details of the code. They do not even need to know that you, and not the SAS Institute, are the author of this clever code.
Figure 7 also includes, at lines 57 through 63, a macro that renames variables P1-P26 to VA-VZ. You should be able to see that the macro evaluates to the statements:

```
Rename
P1 = VA
P2 = VB
P3 = VC
P26 = VZ
```

Your homework assignment is to make %RENAME into a general purpose macro that can be used by passing the names of the two lists as parameters. (Hint: you can use %DIM inside of %RENAME to determine the lengths of the lists.)

In conclusion, this example has been offered as an illustration of how you can use Macro code to make your programs more flexible and useful. With a little practice, the concepts demonstrated here should provide to be a framework on which you can construct user friendly and "intelligent" applications code.

```
Figure 7

NOTE: SAS OPTIONS SPECIFIED ARE:
      WORK: LIBRARY STH.l
1       INCLUDE NLS/NOCLASS;
9       DIMNX YARS: * LIST OF VARIABLES TO BE COUNTED;
10      V VY VZ VX VY VZ VX VY VZ
11      VX VZ VX VZ VX VY VX VY
12      VY VY VX VY VY VY VY
13      DCONM YARS:  
14      THE NUMBER OF ITEMS IN YARS IS 36
15      04LEN\ENy,48
17      DATA ONE:
18      19      DATA TWO:
20      21      DATA THREE:
22      23      DATA FOUR:
24      25      DATA FIVE:
26      27      DATA SIX:
28      29      DATA SEVEN:
30      31      DATA EIGHT:
32      33      DATA NINE:
34      35      DATA TEN:
36      37      DATA ELEVEN:
38      39      DATA TWELVE:
40      41      DATA THIRTEEN:
42      43      DATA FOURTEEN:
44      45      DATA FIFTEEN:
46      47      DATA SIXTEEN:
48      49      DATA SEVENTEEN:
50      51      DATA EIGHTEEN:
52      53      DATA NINETEEN:
54      55      DATA TWENTY:
56      57      DATA TWENTYONE:
58      59      DATA TWENTYTWO:
60      61      DATA TWENTYTTHREE:
62      63      DATA TWENTYFOUR:
64      65      DATA TWENTYFIVE:
66      67      DATA TWENTY_SIX:
```

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

Frederick Pratter
Abt Associates Inc.
33 Wheeler Street
Cambridge, MA 02138
617/492-7100