Wholly MACRO! A handful of techniques to simplify your SAS code and make it recyclable
Christianna S. Williams, Yale University, New Haven, CT

ABSTRACT
Those ampersands and percent symbols can be intimidating to those new to the SAS® Macro Language! However, users whose data manipulation or statistical tasks are ever repetitive or who want their SAS programs to be more flexible and data driven can benefit from introducing some Macro techniques into their code. The purpose of this paper is to provide a straightforward introduction to SAS Macro Language through a series of concrete examples. In each case I will explain the task, show how it could be achieved without Macros, and then demonstrate a Macro technique that will simplify the program and/or make it more widely applicable. The simplest examples will include the use of the %LET statement and automatic macro variables, while the more complex tasks will demonstrate techniques such as %DO loops and the use of CALL SYMPUT to communicate between the DATA STEP and the Macro facility.

INTRODUCTION
Macro programming provides some very powerful tools to make your SAS programs more concise and versatile. Once one is familiar with the primary macro statements and functions and knows a bit about the way they are processed, using macros can also make programs more readable. Many of these statements and functions look much like some used in the data step, and there are similarities, but there are also some important differences.
This paper is designed to hit some of the highlights of what macros can accomplish for users with some SAS programming experience but little or no familiarity with SAS macros. It is my objective to make the reader more comfortable with understanding and writing SAS programs that utilize macros by showing how similar output can be achieved both with and without macros. In most cases, the program with macros will be shorter and more easily adapted to new tasks. Programs containing macros may take longer to develop initially than comparable programs without macros; however, it is my experience that this extra start-up time is more than compensated by time saved later in not having to make as many changes to programs when they are re-used and recycled.
Several of the examples utilize a fictitious data set of hospital admissions, with demographic information for each patient, dates of admission and discharge, the total charges for the admission, and up to ten diagnoses and ten procedure codes per admission. The output of PROC CONTENTS for this data set called EX.HOSPADMIT is provided at the end of the paper.

EXAMPLE: PUTTING A DATE INTO A TITLE OR FOOTNOTE
By default the time and date of the current SAS session will come out in the first title line of output, but that may not be where you want it or in the format you like. Let’s say instead you’d like to have the date in the first footnote. Without using a macro variable or function, one pretty much has to resort to typing the date into the footnote as follows:

```
OPTIONS nocenter nodate;
FOOTNOTE 'Program run on 14JUN01';
```

Whatever the SAS output, it will then include the following at the bottom of the page.

```
Program run on 14JUN01
```

Of course, the disadvantage of this is that one must remember to change the footnote each time the program is run. One of the most useful automatic SAS macro variables is &SYSDATE, which contains the date that the current SAS session was invoked as a SAS date in the DATE7. format, (displays a 2-digit day of the month, the first three letters of the month name, and a 2-digit year. So, changing the above footnote statement to the following:

```
FOOTNOTE "Spiffy new program run on &SYSDATE";
```

will produce the footnote below:

```
Spiffy new program run on 14JUN01
```

Note that it causes no problem to include spaces, other text and punctuation in between the macro variable references as needed. But being limited to these formats may still not be satisfactory; it’d be nice to be able to write the date and time in any format you choose. The %SYSFUNC macro function allows this. Let’s say

```
FOOTNOTE "Started at &SYSTIME on &SYSDAY, &SYSDATE9";
```

produces the following:

```
Started at 09:49 on Thursday, 14JUN2001
```

Simple as this is, it demonstrates something fundamental about macro variables – they are text, NOT numbers or dates. When the SAS macro processor sees a macro variable reference (usually identified by an ampersand), it substitutes the text that is the value of that macro variable. However, if a macro variable is within single quotes, the macro processor doesn’t “see” it, and it remains as text. The macro processor does see – and attempt to resolve – macro variable references enclosed in double quotes.
Getting back to our example, what if you don’t like the DATE7. format? Well, there is another macro variable &SYSDATE9 that will write the current date in DATE9. format (i.e. same as the DATE7. format but with a four-digit year). There are also automatic macro variables &SYSDAY and &SYSTIME which contain the day of the week and the time (in TIME5. format) a SAS job began executing. Putting these together one has some flexibility, for example:

```
FOOTNOTE "Started at &SYSTIME on &SYSDAY, &SYSDATE9";
```

produces the following:

```
Started at 09:49 on Thursday, 14JUN2001
```

Note that it causes no problem to include spaces, other text and punctuation in between the macro variable references as needed.

Christianna S. Williams, Yale University, New Haven, CT
we want to have the time with no leading zeroes and with a.m or p.m. and we want the date written as the day of the month, the month-name, and the four-digit year. The statement:

```
FOOTNOTE
"It is now %SYSFUNC(TIME(),TIMEAMPM.) on %SYSFUNC(DATE(),WORDDATX.);"
```

yields this footnote:

```
It is now 11:56:51 AM on 14 June 2001
```

The % indicates that a macro function (or the name of a macro program) follows. What %SYSFUNC does is allow the execution of a SAS function (the first argument) given the appropriate arguments of that SAS function and then return the result in the format provided as the optional second argument. In this case the SAS functions used are the TIME and DATE functions, which themselves have only the null argument (). Note that not all SAS functions can be used with %SYSFUNC, so it is a good idea to check the documentation, but it is a very useful macro function to get to know.

I'm still not happy with our footnote – those leading blanks before the date are annoying. We can get rid of them by using another call to %SYSFUNC to execute the function LEFT:

```
FOOTNOTE "Now we're cookin' at %SYSFUNC(TIME(),TIMEAMPM.) on %SYSFUNC(LEFT(%SYSFUNC(DATE()), WORDDATX.));"
```

This generates the much nicer footnote:

```
Now we're cookin' at 12:16 PM on 14 June 2001
```

**EXAMPLE: USING %LET TO ASSIGN A VALUE TO A USER-DEFINED MACRO VARIABLE**

Above we saw several examples of automatic macro variables. Automatic macro variables are defined by the SAS system whenever SAS is invoked, and they mainly keep track of information about the SAS session, such as the time and date and the version of SAS being run. Users can reference them almost anywhere in a SAS program except data lines.

However, the SAS macro language facility would not be nearly so powerful if users could not define and assign values to macro variables themselves. User-defined macro variables can be used for an enormous array of purposes. The rest of this paper will demonstrate some of these different applications.

There are several different ways to assign a value to a user-defined macro variable. Perhaps the simplest is the %LET statement. Let's say that this week you wish to report on all the hospital admissions that occurred during June, 2000. You need to tabulate the total and median charges by gender. Next week you'll need to run the exact same report for July. This is a possible program.

```
TITLE1 'Hospital Stays for June 2000 by Gender';
PROC TABULATE DATA=ex.hospitals ;
WHERE admyear = 2000 AND admmonth = 'June';
CLASS gender ;
VAR totcharge ;
TABLES (gender=' ' ALL='Total'),
   (totcharge*(N*F=4. SUM*F=DOLLAR9. MEDIAN*F=DOLLAR7.)) )
/R=12 ROW=FLOAT;
FORMAT gender sexf. ;
LABEL totcharge='Charges: June 2000';
RUN;
```

For this fictitious data set, the following output would be produced:

```
<table>
<thead>
<tr>
<th></th>
<th>Sum</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Male</td>
<td>$69,624</td>
<td>$5,995</td>
</tr>
<tr>
<td>Female</td>
<td>$213,410</td>
<td>$8,067</td>
</tr>
<tr>
<td>Total</td>
<td>$283,034</td>
<td>$7,774</td>
</tr>
</tbody>
</table>
```

To run this again for July would require changes in the title, the WHERE statement and the label for the TOTCHARGE variable. This is not so bad, but what if this TABULATE was only a small portion of the analysis you needed to do each month. Instead, we can create macro variables for the month and year. This could greatly reduce the number of modifications needed for each run of the analysis:

```
%LET month=June;
%LET year=2000;
TITLE1 "Hospital Stays for %month %year by Gender";
```

The output is identical to that shown above. The %LET statements define macro variables &month and &year, which are given (for this iteration) the values 'June' and '2000'. Whenever the macro processor encounters a reference to either of these macro variables, the appropriate text is substituted. Each time the analysis needs to be done for a different month and/or year, one just changes the values of the macro variables in the two %LET statements. No other changes are required. Some macro statements including some described below can only be used within a macro; however, %LET can be used either within a macro or in "open code" (i.e. not within a macro).

It is important to remember that %LET is not the same as an assignment statement within a DATA step in one very important respect – macro variables are always text, even if the value is a number. For example, the statement

```
%LET year=2000+1;
```

assigns the value '2000+1' to the macro variable &YEAR not 2001. In the program above this will not cause an error and will produce the correct results for 2001 because the arithmetic will be carried out when the WHERE statement is executed; however, the title and label may not be as desired. The following output would be produced.

```
```

```
```
Hospital Stays for June 2000+1 by Gender

<table>
<thead>
<tr>
<th></th>
<th>Charges: June 2000+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>----------+----+---------+-------</td>
<td></td>
</tr>
<tr>
<td>Male</td>
<td>10</td>
</tr>
<tr>
<td>Female</td>
<td>21</td>
</tr>
<tr>
<td>Total</td>
<td>31</td>
</tr>
</tbody>
</table>

In order to do arithmetic in a %LET statement, use the %EVAL or %SYSEVALF macro functions. %EVAL will do integer arithmetic; %SYSEVALF will do floating point arithmetic. So, for example:

\%

\LET year=\%EVAL(2000+1) ;

would produce the desired result with the correct title and labels for the year 2001.

Now, this is clearly a silly example – why not just set the &YEAR to 2001? However, the value assigned to a macro variable in a %LET statement can itself include one or more macro variables, and it is in these cases that doing arithmetic in a %LET statement can be more useful. Suppose we want to tabulate the average length of stay for several years and wish to be able to vary the length of the interval. We could do this as follows:

\%

\LET year1=1997;
\LET interval=3;
\LET year2=\%EVAL(year1 + interval);
\TITLE1 "Hospital Stays between &year1 and &year2"
PROC TABULATE DATA=ex.hospitals;
WHERE &year1 LE admyear LE &year2;
CLASS admyear;
VAR los;
TABLES (admyear=' ') ,
(los*(N*F=4. MEAN*F=6.2 STD*F=6.2)) ;
/RTS=12 ROW=FLOAT
BOX="Interval &year1-&year2";
LABEL los='Length of stay';
RUN;

This would produce the following table:

<table>
<thead>
<tr>
<th>Interval</th>
<th>Length of stay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997-2000</td>
<td>N</td>
</tr>
<tr>
<td>1997</td>
<td>6</td>
</tr>
<tr>
<td>1998</td>
<td>207</td>
</tr>
<tr>
<td>1999</td>
<td>270</td>
</tr>
<tr>
<td>2000</td>
<td>267</td>
</tr>
</tbody>
</table>

The very useful %PUT writes the current value of &mysong to the log:

mysong=I just want to bang on a drum all day.

The interior spaces are preserved; however, the leading blanks are removed. To retain them, enclose the text in the %STR function.

\%

\LET mysong=%STR( I just want to bang on a drum all day);
\%PUT mysong=&mysong;
\%

does the trick. In the log, we get

mysong= I just want to bang on a drum all day.

The %STR function removes the meaning from special characters that are known when the macro is compiled. This includes semicolons but not ampersands or percent signs. Thus, with the %STR function, several SAS statements could represent the value of a single macro variable. For example,

\%

\LET check=%STR(PROC PRINT DATA=&SYSLAST;
RUN;) ;

would store this PROC PRINT step as the value of the macro variable &CHECK. It would cause the most recently created data set (denoted by the automatic macro variable &SYSLAST) to print whenever a reference to &CHECK is encountered in the program. This particular shorthand can be useful during debugging when one wants to print the most recent data set numerous times.

One final note about %LET and the %STR function. If the text being assigned to a macro variable contains an unbalanced quotation mark (for example, an apostrophe), take advantage of the fact that the characters immediately following a percent sign are not quoted:

\%

\LET mysong=
%STR(I don't want to work;  I just want to play; I just want to bang on a drum all day!);
\%PUT mysong=&mysong;
\%

will put these lines into the log, apostrophe and all.

EXAMPLE: PASSING PARAMETERS TO A MACRO PROGRAM

It's clear from the above that you can accomplish a lot with just the %LET statement and a few functions, but there is much more to the SAS macro language. This example introduces the first macro program. Returning to the hospitals data set, let's say that you need to generate the table of charges for each year and for several different classification variables – say for gender, age group, and marital status. The following program would produce the table for a single year (1998) and classification variable (age).

\%

\TITLE1 "1998 Hospital Stays by Age ";
PROC TABULATE DATA=ex.hospitals;
WHERE admyear = 1998;
CLASS age;
VAR totcharge;
TABLES (age=' ' ALL='Overall') ,
(totcharge*(N*F=4. SUM='Total'*F=DOLLAR11.));

The value of the macro variable assigned in a %LET statement can be a long string of text. For example, the following is perfectly valid SAS code:

\%

\LET mysong= I just want to bang on a drum all day;
\%PUT mysong=&mysong;
\%

EXAMPLE: PASSING PARAMETERS TO A MACRO PROGRAM

It's clear from the above that you can accomplish a lot with just the %LET statement and a few functions, but there is much more to the SAS macro language. This example introduces the first macro program. Returning to the hospitals data set, let's say that you need to generate the table of charges for each year and for several different classification variables – say for gender, age group, and marital status. The following program would produce the table for a single year (1998) and classification variable (age).

\%

\TITLE1 "1998 Hospital Stays by Age ";
PROC TABULATE DATA=ex.hospitals;
WHERE admyear = 1998;
CLASS age;
VAR totcharge;
TABLES (age=' ' ALL='Overall') ,
(totcharge*(N*F=4. SUM='Total'*F=DOLLAR11.));

The very useful %PUT writes the current value of &mysong to the log:

mysong=I just want to bang on a drum all day.

The interior spaces are preserved; however, the leading blanks are removed. To retain them, enclose the text in the %STR function.

\%

\LET mysong=%STR( I just want to bang on a drum all day);
\%PUT mysong=&mysong;
\%

does the trick. In the log, we get

mysong= I just want to bang on a drum all day.

The %STR function removes the meaning from special characters that are known when the macro is compiled. This includes semicolons but not ampersands or percent signs. Thus, with the %STR function, several SAS statements could represent the value of a single macro variable. For example,

\%

\LET check=%STR(PROC PRINT DATA=&SYSLAST;
RUN;) ;

would store this PROC PRINT step as the value of the macro variable &CHECK. It would cause the most recently created data set (denoted by the automatic macro variable &SYSLAST) to print whenever a reference to &CHECK is encountered in the program. This particular shorthand can be useful during debugging when one wants to print the most recent data set numerous times.

One final note about %LET and the %STR function. If the text being assigned to a macro variable contains an unbalanced quotation mark (for example, an apostrophe), take advantage of the fact that the characters immediately following a percent sign are not quoted:

\%

\LET mysong=
%STR(I don’t want to work;  I just want to play; I just want to bang on a drum all day!);
\%PUT mysong=&mysong;
\%

will put these lines into the log, apostrophe and all.

mysong= I don’t want to work;  I just want to play;  I just want to bang on a drum all day!
The resulting table is below.

<table>
<thead>
<tr>
<th>AGE</th>
<th>1998 Total Charges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td>70-74</td>
<td>32</td>
</tr>
<tr>
<td>75-79</td>
<td>58</td>
</tr>
<tr>
<td>80-84</td>
<td>73</td>
</tr>
<tr>
<td>85+</td>
<td>44</td>
</tr>
<tr>
<td>Overall</td>
<td>207</td>
</tr>
</tbody>
</table>

For four years and three classification variables, one would have to copy and paste and modify the above code 12 times to produce all the desired tables. As an aside, note that while it might be possible to combine several of these into one PROC TABULATE step by including multiple classification variables in a single table, this is not always possible or desirable. For example, if some of the observations have missing data for some classification variables but not others, these observations would be entirely excluded, even from the tabulation for the classification variables that were not missing. Further, PROC TABULATE is just being used as an example procedure here; the same macro techniques could be applied to many other procedures, where it might be impossible or inappropriate to analyze several variables simultaneously.

With a macro, we can take advantage of the fact that the majority of the code for the 12 PROC TABULATEs would be the same with only a few key elements changing between repetitions. These elements that change will be our macro variables and will be passed to the macro program as parameters. In this example, we’d like to vary the year, the CLASS variable, and perhaps the format and title text associated with each classification variable.

Here is the macro code:

```sas
%MACKRO tabit(year,classvar,clstext,clsfmt);
TITLE1 "&year Hospital Stays by &clstext ";
PROC TABULATE DATA=ex.hospitals ;
WHERE admyear = &year ;
CLASS &classvar ;
VAR totcharge ;
TABLES (&classvar=' ' ALL='Overall'),
   (totcharge*(N*F=4. SUM='Total'*F=DOLLAR11. MEDIAN*F=DOLLAR7.))
/RTS=12 ROW=FLOAT box='AGE';
FORMAT &classvar &clsfmt. ;
LABEL totcharge = "&year Total Charges";
RUN;
%MEND tabit;
```

The macro program begins with the %MACKRO statement and ends with the %MEND statement. The %MACKRO statement also includes a user-defined name for the macro program (obeys naming rules for a SAS variable), followed by the macro parameters, separated by commas. These parameters define macro variables, which will be assigned values at the time the macro is invoked (or “called”). In this particular example all the parameters are “positional parameters”: this means it is the position of the parameter within the parenthesis (first, second, third and so on) that indicates which parameter it is.

Until the macro is called, it does not run. Below are the twelve macro calls for the twelve TABULATEs.

```sas
%tabit(1998,gender,Gender,sexf.) ;
%tabit(1999,gender,Gender,sexf.) ;
%tabit(2000,gender,Gender,sexf.) ;
%tabit(1998,age,5-year Age Groups,agef.) ;
%tabit(1999,age,5-year Age Groups,agef.) ;
%tabit(2000,age,5-year Age Groups,agef.) ;
%tabit(1998,marstat,Marital status,msf.) ;
%tabit(1999,marstat,Marital status,msf.) ;
%tabit(2000,marstat,Marital status,msf.) ;
%tabit(2001,marstat,Marital status,msf.) ;
```

The output for three selected tables is reproduced below:

<table>
<thead>
<tr>
<th>2000 Hospital Stays by Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gender</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Male</td>
</tr>
<tr>
<td>Female</td>
</tr>
<tr>
<td>Overall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1998 Hospital Stays by 5-year Age Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-year Age Groups</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>70-74</td>
</tr>
<tr>
<td>75-79</td>
</tr>
<tr>
<td>80-84</td>
</tr>
<tr>
<td>85+</td>
</tr>
<tr>
<td>Overall</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1998 Hospital Stays by Marital status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marital status</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Married</td>
</tr>
<tr>
<td>Divorced</td>
</tr>
<tr>
<td>Widowed</td>
</tr>
<tr>
<td>Never married</td>
</tr>
<tr>
<td>Overall</td>
</tr>
</tbody>
</table>

Unlike the keyword parameters, which are used in a later example, positional parameters do not have default values. Hence, if one is left out, it will be null. This may or may not cause a problem. In this example, if any of the parameters except &CLSFM are not specified, an error will be generated because a
null value will produce an invalid SAS statement when text substitution occurs. However, if one desired the classification variable not to be formatted, the last parameter could be left blank. The following call to the %TABIT macro:

```sas
%tabit(1999,age,Age,);
```

would produce the following output (in part). It made sense to change the value of the &CLSTXT parameter as well.

<table>
<thead>
<tr>
<th>Age</th>
<th>1999 Total Charges</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Total</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>72</td>
<td>12</td>
</tr>
<tr>
<td>73</td>
<td>13</td>
</tr>
<tr>
<td>74</td>
<td>25</td>
</tr>
</tbody>
</table>

One slight modification to the macro could allow one to optionally not specify the &CLSTXT parameter. If it is left blank, we'd like to substitute the variable name. This could be accomplished by adding the following statement:

```sas
%IF %LENGTH(&clstxt) = 0 %THEN %LET clstxt = &classvar;
```

The %IF-%THEN statement allows the conditional execution of the %LET. The condition is based on the length of the value of the &CLSTXT parameter, returned by the %LENGTH macro function. If the &CLSTXT parameter is null, it will have a length of 0.

**EXAMPLE: USING A %DO LOOP**

The fact that one of the parameters we are using is numeric and sequential lends this task nicely to the use of a %DO loop. We define a slightly different macro:

```sas
%MACRO tabit2(classvar=gender, clstxt=, clsfmt=, year1=1998, year2=2001);
%IF %LENGTH(&clstxt) = 0 %THEN %LET clstxt = &classvar;
%DO year = &year1 %TO &year2 ;

TITLE1 "&year Hospital Stays by &clstxt";
PROC TABULATE DATA=ex.hospitals ;
WHERE admyear = &year ;
CLASS &classvar ;
VAR totcharge ;
TABLES ( &classvar = ' ALL'='Overall'),
(totcharge*='N*F=4,' SUM='Total'*='F=DOLLAR11,
MEDIAN='F=DOLLAR7.)) ;
/RTS=12 ROW=FLOAT BOX="&clstxt";
FORMAT &classvar &clsfmt. ;

%END;
%MEND tabit2;
```

In the first call to %TABIT2, no value for &CLASSVAR is specified; so, the default value (gender) is used. Values for the other parameters are specified but the order is different than in the %MACRO statement. This is ok for keyword parameters, since the use of the parameter names in the macro call keeps them straight. You can use a combination of positional and keyword parameters in a %MACRO statement; sometimes this is done when some parameters (the positional ones) are required and others (the keyword parameters) are optional. However, all the positional parameters must precede the keyword parameters.

In the second call to %TABIT2 above, no values for &YEAR1 or &YEAR2 are specified so the code within the %DO loop will execute for all years between the default values of these parameters (1998 to 2001). In the third %TABIT2 invocation, &YEAR2 is specified but not &YEAR1, and the &YEAR2 value provided is the same as the default value of &YEAR1. This is fine—the %DO loop code will execute once, for the &YEAR 1998.

**EXAMPLE: ANOTHER WAY TO DO THE %DO**

It also works to define another macro that is simply the loop which can repetitively call the first %TABIT macro:

```sas
%MACRO loop(year1=1998, year2=2001);
%DO year=%year1 %TO %year2;
%tabit(&year,gender,,sexf.);
%tabit(&year,age,Age Groups, agef.);
%tabit(&year,marstat,Marital Status, msf.,year2=1998);
%END;
%MEND loop;
```

For the default values of the year range, call the %LOOP macro like this:

```sas
%loop();
```

This example illustrates that a parameter passed to a macro can itself be a macro variable. It will be resolved to its current value.
when the %TABIT macro is called by the %DO loop. This method has the advantage of making the entire program more modular, and might be preferable if the same year range was desired for all the classification variables. It would be relatively straightforward to add in other processing that one needed to do for the same year range.

**EXAMPLE: %DO LOOP MACRO IN A DATA STEP**

Sometimes one needs to do the same type of processing to numerous variables. In a DATA step this can often be accomplished with ARRAYs. However, there are some repetitive tasks for which an ARRAY is not well suited, such as code that while it must be done for a bunch of variables, is only executed once – like RENAME, ATTRIB, DROP or KEEP.

In this particular example, let’s say we want to rename the ten diagnosis variables and the ten procedure variables. Currently, they are named DIAG1 - DIAG10 and PROC1 - PROC10. We wish to rename them as DIAG1_NEW – DIAG10_NEW and PROC1_PROCEDURE – PROC10_PROCEDURE. The non-macro code to do this is straightforward but tedious.

```sas
DATA temp1;
SET ex.hospitals;
RENAME
diag1=diag1_new proc1=proc1_new
    diag2=diag2_new proc2=proc2_new
    diag3=diag3_new proc3=proc3_new
    diag4=diag4_new proc4=proc4_new
    diag5=diag5_new proc5=proc5_new
    diag6=diag6_new proc6=proc6_new
    diag7=diag7_new proc7=proc7_new
    diag8=diag8_new proc8=proc8_new
    diag9=diag9_new proc9=proc9_new
    diag10=diag10_new proc10=proc10_new
RUN;
```

Of course, the RENAME could also be implemented as a data set option on the SET statement. A PROC CONTENTS could show that the desired renaming has been achieved.

A simple %DO loop within a macro could do the same thing. The macro RENAMEVAR follows:

```sas
%MACRO renamevar;
RENAME
    diag&n=diag&n._new
    proc&n=proc&n._procedure
%END;
```

For the first time we see amperands in the middle of variable names. Remember that all the macro is doing is text substitution. When the &N is encountered, SAS inserts the current value of that macro variable – in this case the current value of the %DO loop index – which produces a SAS variable name. The SAS code that is generated by the execution of the macro is thus identical to the rename statement we typed out above.

There a few points to note here. First, the code that does not need to be repeated at each iteration of the %DO loop, namely the RENAME keyword and the semi-colon that terminates the RENAME statement, are outside the loop. Second, we see that when the macro variable name is not at the end of the SAS variable (as in DIAG&N_NEW and &N.PROCEDURE above) the end of the macro variable name is indicated by a period. Otherwise, SAS would think the macro variable names were &N.NEW and &N.PROCEDURE, which are not defined, and an error would be generated.

This macro would then be called within the DATA step. All the macro is doing is generating a RENAME statement; thus, as with any RENAME statement, the call can be anywhere within the DATA step. For example,

```sas
DATA temp;
SET ex.hospitals;
__valid SAS statements ... RENAME__valid SAS statements ...
%renamevar
__more valid SAS statements__ ...
RUN;
```

The call to the RENAMEVAR macro itself does not need to have a semicolon at the end, since the semi-colon ending the RENAME statement the macro writes is within the macro.

There is another way to do this that is slightly more general. We write a new macro %RENAME2.

```sas
%MACRO rename2(oldpfx,newpfx,
    start=1,finish=10);
RENAME
    &oldpfx&o=n=&newpfx&o=n
%END;
```

With this macro we could rename an arbitrary number of variables as long as both the old names and the new names were of the form ABCDn and XYZn. The first two macro parameters are positional parameters, are required and specify the “prefixes” of the old variable names and the new variable names respectively. The other two parameters are keyword parameters and indicate the start and stop points for the index of the %DO loop. Here they are given default values of 1 and 10 respectively.

Hence, to rename all the DIAGn variables to DIAGNOSIn and the PROCn variables to PROCEDUREn, we would simply call the %RENAME2 macro twice in our DATA step. A single RENAME statement would be generated for each call to the macro. Here, I am just renaming the first five procedure variables, but this would of course also work in a situation where different variables in need of renaming were repeated different numbers of times.

```sas
DATA temp2;
SET ex.hospitals;
%rename2(diag,diagnosis)
    %rename2(proc,procedure,start=1,end=5)
RUN;
```

Now, it might seem a little silly to write a macro to just rename fifteen or twenty variables, but I have had jobs where I needed to rename hundreds of variables, and this technique has come in handy. One situation in which it is particularly useful is if you have multiple data sets that are similarly organized from different time periods (or different sites). You need to join them into a single data set – perhaps to examine changes over time. The data sets have many variables with the same names, which need to be renamed to indicate the source (e.g. add a year suffix) and to prevent information from one data set overwriting that from another. Specifically, let’s say you have ten data sets named OLD1990 – OLD2000. Each has 50 variables named A1-A10, B1-B20, C1-C5, D1-D10, E1-E5. You wish to merge these data sets, and rename each variable as PREFIX-YEAR_NUMBER. For example variable B5 from 1996 would become B1996_5. We would also like to modify the LABEL for each variable to contain the year. An abbreviated PROC CONTENTS for the old and new datasets for 1990 are reproduced at the end of the paper. The
following macro could help reduce your typing somewhat:

```sas
%MACRO rename3(prefix, year, start=1, finish=10);
RENAME
%DO n = &start %TO &finish;
  &prefix&n = &prefix&year._&n
%END;
LABEL
%DO n = &start %TO &finish;
  &prefix&n = "Scale &prefix, Year &year, Item &n"
%END;
%MEND;
```

Following the MACRO code would be a DATA step for each year, containing a call to the %RENAME3 macro for each set of variables:

```sas
DATA new1990;
  SET old1990;
  %rename3(a, 1990)
  %rename3(b, 1990, finish=20)
  %rename3(c, 1990, finish=5)
  %rename3(d, 1990)
  %rename3(e, 1990, finish=5)
RUN;

DATA new1991;
  SET old1991;
  %rename3(a, 1991)
  %rename3(b, 1991, finish=20)
  %rename3(c, 1991, finish=5)
  %rename3(d, 1991)
  %rename3(e, 1991, finish=5)
RUN;
...
and so on...
```

The need to repeat the DATA step 10 times suggests that this could be a macro as well.

```sas
%MACRO eachyear(year);
DATA new&year;
  SET old&year;
  %rename3(a, &year)
  %rename3(b, &year, finish=20)
  %rename3(c, &year, finish=5)
  %rename3(d, &year)
  %rename3(e, &year, finish=5)
RUN;
%MEND eachyear;
```

The %EACHYEAR macro would then be called once for each year, as follows:

```sas
%eachyear(1990);
%eachyear(1991);
...
%eachyear(2000);
```

As long as you keep your macro variable references straight, it is no problem for one macro to contain calls to another previously defined macro. Once we've gotten this far it makes sense to make a loop for the years as well since they change in a regular fashion also.

```sas
%MACRO yearloop(firstyear, lastyear);
%DO yr = &firstyear %TO &lastyear;
  DATA new&yr;
    SET old&yr;
    %eachyear(&yr)
  %END;
%MEND;
```

A single call to this macro does the whole job:

```sas
%yearloop(1990, 2000);
```

Since we already have the %EACHYEAR macro, we could use it inside the %YEARLOOP macro as well:

```sas
%MACRO yearloop(firstyear, lastyear);
  %DO yr = &firstyear %TO &lastyear;
    %eachyear(&yr)
  %END;
%MEND;
```

Your program is now much shorter and much more modular. It contains three macro definitions and one macro call, which altogether accomplish a lot of repetitive work. Every job is a little bit different, but this general technique can find wide application. This example also illustrates two points about the development and use of macros. First, one needs to identify the repetitive elements and exploit those in defining the tasks that are suited to macros. This can be done progressively, as we have with our renaming task. Second, when DATA set names and variable names are being selected, it can pay off to think about the kinds of tasks that may need to be done later and assign names that may lend themselves to this type of repetitive processing.

**EXAMPLE: USING CALL SYMPUT TO PASS INFORMATION FROM A DATA STEP TO THE MACRO FACILITY**

There are times when you would like to have information from a DATA step available in other parts of a program – for example the value of one or more variables or data set attributes. One way to pass this information is by storing it in macro variables, which can be resolved elsewhere in the program. The CALL SYMPUT routine is one very handy way to take a value from a DATA step and assign it to a macro variable. As usual, a few examples will help illustrate.

Let's say you would like to determine what is the lowest and the highest value of the average daily charges for the hospitalization data set because you'd like to compare this to data you have from other hospitals. Several SAS procedures can provide this information. Here is the code for using PROC UNIVARIATE.

```sas
PROC UNIVARIATE DATA=ex.hospadmit NOPRINT;
VAR chgperday;
OUTPUT OUT=minmax MIN=minimum MAX=maximum;
RUN;
PROC PRINT DATA=minmax noobs;
RUN;
```

This output data set contains a single observation. Thus, we get the following listing:

```
maximum    minimum
1054.43     768.75
```

If we want to use this data elsewhere in our program – for example to locate values within 5% of these extremes in other data sets, we could merge this information in to the other data sets to allow comparison, but making these into macro variables could provide more flexibility. CALL SYMPUT will accomplish this:

```sas
%rename3(a, &year)
%rename3(b, &year, finish=20)
%rename3(c, &year, finish=5)
%rename3(d, &year)
%rename3(e, &year, finish=5)
RUN;
%MEND;
```

A single call to this macro does the whole job:

```sas
%yearloop(1990, 2000);
```

Since we already have the %EACHYEAR macro, we could use it inside the %YEARLOOP macro as well:

```sas
%MACRO yearloop(firstyear, lastyear);
  %DO yr = &firstyear %TO &lastyear;
    %eachyear(&yr)
  %END;
%MEND;
```

Your program is now much shorter and much more modular. It contains three macro definitions and one macro call, which altogether accomplish a lot of repetitive work. Every job is a little bit different, but this general technique can find wide application. This example also illustrates two points about the development and use of macros. First, one needs to identify the repetitive elements and exploit those in defining the tasks that are suited to macros. This can be done progressively, as we have with our renaming task. Second, when DATA set names and variable names are being selected, it can pay off to think about the kinds of tasks that may need to be done later and assign names that may lend themselves to this type of repetitive processing.

**EXAMPLE: USING CALL SYMPUT TO PASS INFORMATION FROM A DATA STEP TO THE MACRO FACILITY**

There are times when you would like to have information from a DATA step available in other parts of a program – for example the value of one or more variables or data set attributes. One way to pass this information is by storing it in macro variables, which can be resolved elsewhere in the program. The CALL SYMPUT routine is one very handy way to take a value from a DATA step and assign it to a macro variable. As usual, a few examples will help illustrate.

Let's say you would like to determine what is the lowest and the highest value of the average daily charges for the hospitalization data set because you'd like to compare this to data you have from other hospitals. Several SAS procedures can provide this information. Here is the code for using PROC UNIVARIATE.

```sas
PROC UNIVARIATE DATA=ex.hospadmit NOPRINT;
VAR chgperday;
OUTPUT OUT=minmax MIN=minimum MAX=maximum;
RUN;
PROC PRINT DATA=minmax noobs;
RUN;
```

This output data set contains a single observation. Thus, we get the following listing:

```
maximum    minimum
1054.43     768.75
```

If we want to use this data elsewhere in our program – for example to locate values within 5% of these extremes in other data sets, we could merge this information in to the other data sets to allow comparison, but making these into macro variables could provide more flexibility. CALL SYMPUT will accomplish this:
DATA _NULL_;  
SET minmax ;  
CALL SYMPUT('mincharge',minimum) ;  
CALL SYMPUT('maxcharge',maximum) ;  
RUN;  
%PUT Minimum daily charge is &mincharge ;  
%PUT Maximum daily charge is &maxcharge ;  

The first CALL SYMPUT statement creates a macro variable &MINCHARGE (note the single quotes) and it is initialized with the value of the DATA step variable MINIMUM. Similarly the second CALL SYMPUT creates a macro variable &MAXCHARGE and assigns it the value of MAXIMUM on the MINMAX data set. There is one important caveat: the macro variables created in this way are not actually assigned their values until the completion of the DATA step. Thus, they cannot be referenced in that same DATA step. However, the values could be resolved in a subsequent PROC or DATA step, and the %PUT statements above will write the following to the log:

Minimum daily charge is 837.53
Maximum daily charge is 101225.28

EXAMPLE: USING CALL SYMPUT TO HELP TRANPOSE A DATA SET
There is another situation in which I frequently find this type of use of CALL SYMPUT to be extremely handy. Let’s say I want to take our hospitalization data set, which has a variable number of observations per person and convert it into a data set with one observation per person with variables for each admission date, discharge date, and total charges. In order to do this I need to know how many admission date, discharge date and total charge variables I need – in other words, I need to know what the maximum number of observations per subject there is in the input data set. There are numerous ways to obtain this information. Here is a method using PROC SUMMARY and PROC MEANS:

PROC SUMMARY DATA=ex.hospadmit NWAY ; 
CLASS studyid ;  
VAR admdate ;  
OUTPUT OUT=numadmit N=numadmits; 
RUN;  

PROC MEANS DATA=numadmit MAX MAXDEC=0; 
VAR numadmits ; 
LABEL numadmits = 'Number of admissions'; 
RUN;  

The SUMMARY step produces no listing but a data set with one observation per STUDYID with a variable NUMADMITS which is the number of observations with nonmissing values for the ADMDATE variable (i.e. the number of admissions). The MEANS step simply prints out the maximum value of this variable as shown below:

<table>
<thead>
<tr>
<th>The MEANS Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Variable : numadmits</td>
</tr>
<tr>
<td>Number of admissions</td>
</tr>
<tr>
<td>Maximum</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

The next step is to incorporate this information into a DATA step that transposes the HOSPADMIT file into one observation per STUDYID:

DATA alladmits ;  
SET ex.hospadmit (KEEP = studyid admdate disdate totcharge);  
BY studyid ;  
ARRAY adm{12} admdate1 - admdate12 ;  
ARRAY dis{12} disdate1 - disdate12 ;  
ARRAY chg{12} totchrg1 - totchrg12 ;  
IF FIRST.studyid THEN DO;  
   DO i = 1 TO 12;  
      adm{i} = . ;  
      dis{i} = . ;  
      chg{i} = . ;  
   END;  
   numadmit = 0 ;  
END;  
RETAIN numadmit admdate1-admdate12 disdate1-disdate12 totchrg1-totchrg12;  
ATTRIB 
   admdate1-admdate12 LENGTH=6 FORMAT=DATE9. LABEL='Admission Date' 
   disdate1-disdate12 LENGTH=6 FORMAT=DATE9. LABEL='Discharge Date' 
   totchrg1-totchrg12 LABEL='Total Charges' numadmit LABEL='# Admits this StudyID' ; 
   numadmit = numadmit + 1 ; 
   adm{numadmit} = admdate ; 
   dis{numadmit} = disdate ; 
   chg{numadmit} = totcharge ;  
IF LAST.studyid THEN OUTPUT ;  
DROP i admdate disdate totcharge ;  
RUN;  

The fact that the number 12 appears in the above code thirteen times all of which would need to be changed (based on a repeat of the SUMMARY steps above) each time the input data changed suggests that it would make this code much more general if we could macro-ize the maximum number of observations. CALL SYMPUT is perfect for this. We replace the PROC MEANS above with another SUMMARY and add a quick DATA _NULL_ with our CALL SYMPUT:

PROC SUMMARY DATA=numadmit ; 
VAR numadmits ; 
OUTPUT OUT=getmax MAX=maxadmits; 
RUN; 

DATA _NULL_;  
SET getmax ;  
CALL SYMPUT('maxadm',LEFT(PUT(maxadmits,2.))); 
RUN; 

We now have a macro variable &MAXADM that has as its value the maximum number of admissions per subject in our HOSPADMIT data set (currently 12). The use of the LEFT and PUT functions in the CALL SYMPUT statement above is primarily cosmetic: it controls the numeric-to-character conversion and eliminates leading blanks. Next we simply repeat our transposing DATA step, substituting a reference to &MAXADM for every instance of a 12. Voila! The program is now much more data-driven!
DATA alladmits;
  SET ex.hospadmit (KEEP = studyid admdate disdate totcharge);
BY studyid;
ARRAY adm[&maxadm] admdate1-admdate&maxadm;
ARRAY dis[&maxadm] disdate1-disdate&maxadm;
ARRAY chg[&maxadm] totchrg1-totchrg&maxadm;
IF FIRST.studyid THEN DO;
  DO i = 1 TO &maxadm;
    adm[i] = .;
    dis[i] = .;
    chg[i] = .;
  END;
  numadmit = 0;
END;
RETAIN numadmit admdate1-admdate&maxadm
          disdate1-disdate&maxadm
totchrg1-totchrg&maxadm;
ATTRIB
  admdate1-admdate&maxadm LENGTH=6
  FORMAT=DATE9.  LABEL='Admission Date'
  disdate1-disdate&maxadm LENGTH=6
  FORMAT=DATE9.  LABEL='Discharge Date'
  tootchrg1-totchrg&maxadm LABEL='Total Charges'
  numadmit LABEL='# Admits this StudyID' ;
  numadmit = numadmit + 1;
  adm[numadmit] = admdate;
  dis[numadmit] = disdate;
  chg[numadmit] = totcharge;
IF LAST.studyid THEN OUTPUT;
DROP i admdate disdate totcharge;
RUN;

CONCLUSIONS
For readers new to SAS macro language, I hope the examples presented in this paper will provide incentive to start using macros in your programming. And for those already familiar with macros, I hope I’ve offered a new technique or two. I have illustrated only a few of the ways that each of these macro tools can be used, but they are among the applications that I have found most useful.

Sometimes using macros or macro variables really is the only way to accomplish a particular task. Often, however, macros are not absolutely required, so the first challenge is to determine when introducing macro techniques into a program will result in an improvement. From my experience there are two general situations when this is the case.

First, when there is a very repetitive aspect to the job, macros can shorten a program tremendously, rendering it easier to read and modify. One needs to identify which elements of the task are constant and which ones change among the repetitions. Code the constant elements into the guts of the macro; assign the variable elements as macro variables, whether that is through %LET statements, the use of macro parameters or both.

Second, when one needs to get information from one place in a program to another, whether it is from a DATA step to a TITLE or from one DATA step to another, macros are often a very flexible solution. These types of applications can often make your code more data-driven – more recyclable! They can often save a lot of programmer time in the long run as well.

CONTACT INFORMATION
Your comments and questions are welcome. Contact the author at Christianna.Williams@yale.edu.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ©indicates USA registration.
### Data Set Name: WORK.NEW1990
- Observations: 100
- Member Type: DATA
- Variables: 51
- Engine: V8
- Indexes: 0
- Release Created: 8.0101M0
- Host Created: WIN_98

#### Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>ID</td>
<td>Num</td>
<td>4</td>
<td>260</td>
<td>Subject ID</td>
</tr>
<tr>
<td>1</td>
<td>a1990_1</td>
<td>Num</td>
<td>3</td>
<td>264</td>
<td>Scale a, Year 1990, Item 1</td>
</tr>
<tr>
<td>2</td>
<td>a1990_2</td>
<td>Num</td>
<td>3</td>
<td>267</td>
<td>Scale a, Year 1990, Item 2</td>
</tr>
<tr>
<td>10</td>
<td>a1990_10</td>
<td>Num</td>
<td>3</td>
<td>291</td>
<td>Scale a, Year 1990, Item 10</td>
</tr>
<tr>
<td>11</td>
<td>b1990_1</td>
<td>Num</td>
<td>8</td>
<td>0</td>
<td>Scale b, Year 1990, Item 1</td>
</tr>
<tr>
<td>29</td>
<td>b1990_19</td>
<td>Num</td>
<td>8</td>
<td>144</td>
<td>Scale b, Year 1990, Item 19</td>
</tr>
<tr>
<td>30</td>
<td>b1990_20</td>
<td>Num</td>
<td>8</td>
<td>152</td>
<td>Scale b, Year 1990, Item 20</td>
</tr>
<tr>
<td>31</td>
<td>c1990_1</td>
<td>Num</td>
<td>3</td>
<td>294</td>
<td>Scale c, Year 1990, Item 1</td>
</tr>
<tr>
<td>35</td>
<td>c1990_5</td>
<td>Num</td>
<td>3</td>
<td>306</td>
<td>Scale c, Year 1990, Item 5</td>
</tr>
<tr>
<td>36</td>
<td>d1990_1</td>
<td>Num</td>
<td>8</td>
<td>160</td>
<td>Scale d, Year 1990, Item 1</td>
</tr>
<tr>
<td>44</td>
<td>d1990_9</td>
<td>Num</td>
<td>8</td>
<td>224</td>
<td>Scale d, Year 1990, Item 9</td>
</tr>
<tr>
<td>45</td>
<td>d1990_10</td>
<td>Num</td>
<td>8</td>
<td>232</td>
<td>Scale d, Year 1990, Item 10</td>
</tr>
<tr>
<td>46</td>
<td>e1990_1</td>
<td>Num</td>
<td>4</td>
<td>240</td>
<td>Scale e, Year 1990, Item 1</td>
</tr>
<tr>
<td>50</td>
<td>e1990_5</td>
<td>Num</td>
<td>4</td>
<td>256</td>
<td>Scale e, Year 1990, Item 5</td>
</tr>
</tbody>
</table>

### Data Set Name: EX.HOSPADMIT
- Observations: 986
- Member Type: DATA
- Variables: 32
- Engine: V8
- Indexes: 0
- Release Created: 8.0101M0
- Host Created: WIN_98

#### Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>StudyID</td>
<td>Num</td>
<td>5</td>
<td>181</td>
<td></td>
<td>respondent ID</td>
</tr>
<tr>
<td>22</td>
<td>admdate</td>
<td>Num</td>
<td>4</td>
<td>16</td>
<td>DATE7.</td>
<td>date of admission</td>
</tr>
<tr>
<td>28</td>
<td>admonth</td>
<td>Char</td>
<td>9</td>
<td>172</td>
<td></td>
<td>Month of admission</td>
</tr>
<tr>
<td>27</td>
<td>admyear</td>
<td>Num</td>
<td>4</td>
<td>28</td>
<td></td>
<td>4-digit Year of admission</td>
</tr>
<tr>
<td>26</td>
<td>age</td>
<td>Num</td>
<td>4</td>
<td>24</td>
<td></td>
<td>Best Baseline Age</td>
</tr>
<tr>
<td>32</td>
<td>agegroup</td>
<td>Num</td>
<td>3</td>
<td>195</td>
<td></td>
<td>Age in 5-year groups</td>
</tr>
<tr>
<td>30</td>
<td>chpperday</td>
<td>Num</td>
<td>8</td>
<td>0</td>
<td></td>
<td>Average daily charges</td>
</tr>
<tr>
<td>1</td>
<td>diag1</td>
<td>Char</td>
<td>7</td>
<td>32</td>
<td>7.</td>
<td>discharge diagnosis 1</td>
</tr>
<tr>
<td>2</td>
<td>diag2</td>
<td>Char</td>
<td>7</td>
<td>39</td>
<td>7.</td>
<td>discharge diagnosis 2</td>
</tr>
<tr>
<td>3</td>
<td>diag3</td>
<td>Char</td>
<td>7</td>
<td>46</td>
<td>7.</td>
<td>discharge diagnosis 3</td>
</tr>
<tr>
<td>4</td>
<td>diag4</td>
<td>Char</td>
<td>7</td>
<td>53</td>
<td>7.</td>
<td>discharge diagnosis 4</td>
</tr>
<tr>
<td>5</td>
<td>diag5</td>
<td>Char</td>
<td>7</td>
<td>60</td>
<td>7.</td>
<td>discharge diagnosis 5</td>
</tr>
<tr>
<td>6</td>
<td>diag6</td>
<td>Char</td>
<td>7</td>
<td>67</td>
<td>7.</td>
<td>discharge diagnosis 6</td>
</tr>
<tr>
<td>7</td>
<td>diag7</td>
<td>Char</td>
<td>7</td>
<td>74</td>
<td>7.</td>
<td>discharge diagnosis 7</td>
</tr>
<tr>
<td>8</td>
<td>diag8</td>
<td>Char</td>
<td>7</td>
<td>81</td>
<td>7.</td>
<td>discharge diagnosis 8</td>
</tr>
<tr>
<td>9</td>
<td>diag9</td>
<td>Char</td>
<td>7</td>
<td>88</td>
<td>7.</td>
<td>discharge diagnosis 9</td>
</tr>
<tr>
<td>10</td>
<td>diag10</td>
<td>Char</td>
<td>7</td>
<td>95</td>
<td>7.</td>
<td>discharge diagnosis 10</td>
</tr>
<tr>
<td>23</td>
<td>disdate</td>
<td>Num</td>
<td>4</td>
<td>20</td>
<td>DATE7.</td>
<td>date of discharge</td>
</tr>
<tr>
<td>24</td>
<td>gender</td>
<td>Num</td>
<td>3</td>
<td>186</td>
<td></td>
<td>Gender</td>
</tr>
<tr>
<td>29</td>
<td>los</td>
<td>Num</td>
<td>3</td>
<td>192</td>
<td></td>
<td>Length of stay (days)</td>
</tr>
<tr>
<td>25</td>
<td>marstat</td>
<td>Num</td>
<td>3</td>
<td>189</td>
<td></td>
<td>Marital Status: 1=M, 2=S, 3=D, 4=W, 5=N</td>
</tr>
<tr>
<td>11</td>
<td>proc1</td>
<td>Char</td>
<td>7</td>
<td>102</td>
<td>7.</td>
<td>procedure code 1</td>
</tr>
<tr>
<td>12</td>
<td>proc2</td>
<td>Char</td>
<td>7</td>
<td>109</td>
<td>7.</td>
<td>procedure code 2</td>
</tr>
<tr>
<td>13</td>
<td>proc3</td>
<td>Char</td>
<td>7</td>
<td>116</td>
<td>7.</td>
<td>procedure code 3</td>
</tr>
<tr>
<td>14</td>
<td>proc4</td>
<td>Char</td>
<td>7</td>
<td>123</td>
<td>7.</td>
<td>procedure code 4</td>
</tr>
<tr>
<td>15</td>
<td>proc5</td>
<td>Char</td>
<td>7</td>
<td>130</td>
<td>7.</td>
<td>procedure code 5</td>
</tr>
<tr>
<td>16</td>
<td>proc6</td>
<td>Char</td>
<td>7</td>
<td>137</td>
<td>7.</td>
<td>procedure code 6</td>
</tr>
<tr>
<td>17</td>
<td>proc7</td>
<td>Char</td>
<td>7</td>
<td>144</td>
<td>7.</td>
<td>procedure code 7</td>
</tr>
<tr>
<td>18</td>
<td>proc8</td>
<td>Char</td>
<td>7</td>
<td>151</td>
<td>7.</td>
<td>procedure code 8</td>
</tr>
<tr>
<td>19</td>
<td>proc9</td>
<td>Char</td>
<td>7</td>
<td>158</td>
<td>7.</td>
<td>procedure code 9</td>
</tr>
<tr>
<td>20</td>
<td>proc10</td>
<td>Char</td>
<td>7</td>
<td>165</td>
<td>7.</td>
<td>procedure code 10</td>
</tr>
<tr>
<td>31</td>
<td>totcharge</td>
<td>Num</td>
<td>8</td>
<td>8</td>
<td></td>
<td>Total charges for admission</td>
</tr>
</tbody>
</table>