10

Useful SAS® Procedures

10.1 Using the SORT Procedure to Eliminate Duplicate Observations

A data set often contains duplicate values across observations. Individual records can be duplicated or a subset of variables among records can have identical values. One way to eliminate duplicate records from a data set is to use FIRST.VARIABLE and LAST.VARIABLE within the DATA step, which was illustrated in Section 4.2.2. If you don't want to alter the composition of your original data set, however, you can use the NODUPKEY or the NODUPRECS options in the SORT procedure and send the output to a second data set that is referenced in the OUT = option. The form for PROC SORT with these options is as follows:

```
PROC SORT <DATA=SAS-data-set>
          <OUT=SAS-data-set>
          <NODUPKEY>
          <NODUPRECS>;
   BY <DESCENDING> variable-1 <... <DESCENDING> variable-n>;
RUN;
```

10.1.1 Eliminating Observations with Duplicate BY Values

Consider the data set DUPLICATES, which contains three variables: ID, GRADE, and SCORE. Depending upon the BY variable being examined, the number of observations with the same BY values varies. For example, if ID is the BY variable, there are four observations (observations 1, 2, 4, and 6) with identical 'A01' values and three with identical 'A02' values.

DUPLICATES:

	ID	GRADE	SCORE
1	A01	A	3
2	A01	B	3
3	A02	A	4

(*Continued*)

	ID	GRADE	SCORE
4	A01	A	3
5	A02	B	2
6	A01	B	2
7	A02	B	2

When the NODUPKEY option in PROC SORT is used, observations with duplicate BY values are eliminated. PROC SORT sorts first and then compares BY values for a given observation to what is recorded in the previous observation. When a match is found, the current observation is not written to the output data set. Program 10.1 illustrates the different results by using different combinations of BY variables in PROC SORT. In the last PROC SORT, the keyword _ALL_ is used to sort all the variables in the input data set.

Program 10.1:

```
libname ch10 'W:\SAS Book\dat';
proc sort data = ch10.duplicates out = by_id_1 nodupkey;
    by id;
run;

proc sort data = ch10.duplicates out = by_id_grade_1 nodupkey;
    by id Grade;
run;

proc sort data = ch10.duplicates out = by_all_1 nodupkey;
    by _all_;
run;


title 'NODUPKEY: by ID';
proc print data = by_id_1;
run;

title 'NODUPKEY: by ID and GRADE';
proc print data = by_id_grade_1;
run;


title 'NODUPKEY: by ID, GRADE, and SCORE';
proc print data = by_all_1;
run;
```

Output from Program 10.1:

```
                NODUPKEY: by ID

          Obs      ID       grade    score

           1       A01        A        3
           2       A02        A        4
```

NODUPKEY: by ID and GRADE

Obs	ID	grade	score
1	A01	A	3
2	A01	B	3
3	A02	A	4
4	A02	B	2

NODUPKEY: by ID, GRADE, and SCORE

Obs	ID	grade	score
1	A01	A	3
2	A01	B	2
3	A01	B	3
4	A02	A	4
5	A02	B	2

10.1.2 Eliminating Duplicate Observations

When using the NODUPRECS or NODUP options, PROC SORT eliminates duplicate observations. PROC SORT compares all variable values for each observation to the ones from the previous observation. If a match is found, then the observation is not written to the output data set.

PROC SORT compares only adjacent observations for eliminating observations. To correctly eliminate all the duplicate observations, all the variables in the input set need to be sorted. In the DUPLICATES data set, there are two identical observations for ID equals 'A01' and two identical observations for 'A02'. Program 10.2 illustrates what happens when different variables are used in the BY statement with the NODUP option. In the first PROC SORT, a duplicate observation is not eliminated because two identical observations for 'A01' do not follow each other after sorting ID. In the second PROC SORT, all duplicate records are eliminated because BY now sorts by ID and within each ID by GRADE. Similarly, sorting all the variables in the last PROC SORT will also eliminate all duplicates. Thus, to guarantee the elimination of duplicate observations from a data set, sort BY _ALL_ and use either the NODUPRECS or NODUP option.

Program 10.2:

```
proc sort data = ch10.duplicates out = by_id_2 nodup;
    by id;
run;

proc sort data = ch10.duplicates out = by_id_grade_2 nodup;
    by id Grade;
run;

proc sort data = ch10.duplicates out = by_all_2 nodup;
    by _all_;
run;
```

```
title 'NODUP: by ID';
proc print data = by_id_2;
run;

title 'NODUP: by ID and GRADE';
proc print data = by_id_grade_2;
run;

title 'NODUP: by ID, GRADE, and SCORE';
proc print data = by_all_2;
run;
```

Output from Program 10.2:

```
                NODUP: by ID
Obs      ID       grade    score
 1      A01         A        3
 2      A01         B        3
 3      A01         A        3
 4      A01         B        2
 5      A02         A        4
 6      A02         B        2

  NODUP: by ID and GRADE
Obs      ID       grade    score
 1      A01         A        3
 2      A01         B        3
 3      A01         B        2
 4      A02         A        4
 5      A02         B        2

NODUP: by ID, GRADE, and SCORE
Obs      ID       grade    score
 1      A01         A        3
 2      A01         B        2
 3      A01         B        3
 4      A02         A        4
 5      A02         B        2
```


10.2 Using the COMPARE Procedure to Compare the Contents of Two Data Sets

Sometimes two data sets need to be compared to see if they are the same or if they are different. Instead of using DATA step programming, you can use the COMPARE procedure to compare the contents of two data sets or

selected variables from different data sets or within the same data sets. The syntax for PROC COMPARE is listed below:

```
PROC COMPARE <BASE=SAS-data-set>
             <COMPARE=SAS-data-set>
             <BRIEFSUMMARY>;
   BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>;
   ID <DESCENDING> variable-1 <...<DESCENDING> variable-n>;
   VAR variable(s);
   WITH variable(s);
RUN;
```

There are a myriad of options available in PROC COMPARE for controlling how values are compared and how reports are generated. Only a few options are introduced in this chapter. Readers should refer to SAS documentation for additional information.

10.2.1 Information Provided from PROC COMPARE

PROC COMPARE compares the *base* data set with the *comparison* data set specified in the BASE= and COMPARE= options, respectively. PROC COMPARE starts with comparing data set attributes, examining whether both data sets contain matching variables (variables with the same variable name), checking attributes of the matching variables, and determining whether both data sets have matching observations. The matching variables must have the same data type. The matching observations either have the same values for variables specified in the ID statement or occur in the same position in the data sets. After making these comparisons, PROC COMPARE compares the values in the portions of the data sets that match.

In Program 10.3, two data sets are created: COMPARE1 and COMPARE2. Four variables in both COMPARE1 and COMPARE2 have the same variable names: STUDENTID, GENDER, HEIGHT, and WEIGHT1. Among these four variables, HEIGHT has character data type in the COMPARE1 data set, but HEIGHT has numeric data type in the COMPARE2 data set. Another difference between these two data sets is that the WEIGHT2 variable exists only in the COMPARE1 data set. Also, variable WEIGHT1 is assigned with numeric 6.2 format and labeled with "first weight value" only in COMPARE1. The last difference between these two data sets is that there are four observations in COMPARE1 but only three observations in COMPARE2.

Program 10.3 uses only the BASE= and COMPARE= options to compare the COMPARE1 and COMPARE2 data sets without utilizing other options. Because the ID statement is not used in the program, values in COMPARE1 and COMPARE2 are compared based on matching variables with the same name and type, and the matching observations occur in the same position

of the data set. That is to say, only values in the first three observations in COMPARE1 and all the observations in COMPARE2 in STUDENTID, GENDER, and WEIGHT1 variables are compared.

In the generated output from PROC COMPARE, the "Variable Summary" table summarizes variable differences, including the number of common variables (4), the number of variables in one data set but not in the other (one in COMPARE1 but not in COMPARE2), the number of variables that have conflicting types (1), and the number of variables that have different attributes (1). The variable with conflicting type (HEIGHT) appears in the "Listing of Common Variables with Conflicting Types" table. The variable with different attributes (WEIGHT1) appears in the "Listing of Common Variables with Differing Attributes" table.

The "Observation Summary" table summarizes observation differences, such as the number of observations in common (3), the number of observations in one data set but not in the other (1), the number of observations that have unequal values in some compared variables (2), the number of observations that have equal values for all compared variables (1), etc.

The "Value Comparison Summary" table lists the number of variables that have equal values across all comparable observations (1), the number of variables that have unequal values with some observations (2), the total number of unequal values (2), and the maximum difference in unequal numerical values (0.1).

The variables with unequal values (STUDENTID and WEIGHT1) are listed in the "Variables with Unequal Values" table. All the values that have differences in the two data sets are listed in the "Value Comparison Results for Variables" table.

Program 10.3:

```
data compare1;
    input studentID $ gender $ height $ weight1 weight2;
    format weight1 6.2;
    label weight1 = first weight value;
    datalines;
A01 M 67 154.3 154.7
A02 M 64 150.2 150.2
A03 F 58 149.1 149.2
A04 F 59 149.2 149.2
;

data compare2;
    input studentID $ gender $ height weight1;
    datalines;
A01 M 67 154.3
A02 M 64 150.3
A04 F 59 149.1
;
```

```
title 'PROC COMPARE: basic output';
proc compare base = compare1
             compare = compare2;
run;
```

Output from Program 10.3:

```
                    PROC COMPARE: basic output
                       The COMPARE Procedure
          Comparison of WORK.COMPARE1 with WORK.COMPARE2
                           (Method = EXACT)
                          Data Set Summary
Dataset                    Created           Modified  NVar  NObs
WORK.COMPARE1   06AUG12:16:41:03   06AUG12:16:41:03     5     4
WORK.COMPARE2   06AUG12:16:41:03   06AUG12:16:41:03     4     3

                        Variables Summary
Number of Variables in Common: 4.
Number of Variables in WORK.COMPARE1 but not in WORK.
   COMPARE2: 1.
Number of Variables with Conflicting Types: 1.
Number of Variables with Differing Attributes: 1.

      Listing of Common Variables with Conflicting Types
       Variable     Dataset            Type     Length
       height       WORK.COMPARE1      Char          8
                    WORK.COMPARE2      Num           8

     Listing of Common Variables with Differing Attributes
       Variable     Dataset            Type     Length  Format
       weight1      WORK.COMPARE1      Num           8  6.2
                    WORK.COMPARE2      Num           8

                      Observation Summary
                Observation       Base    Compare
                First Obs         1       1
                First Unequal     2       2
                Last Unequal      3       3
                Last Match        3       3
                Last Obs          4       .

Number of Observations in Common: 3.
Number of Observations in WORK.COMPARE1 but not in
WORK.COMPARE2: 1.
Total Number of Observations Read from WORK.COMPARE1: 4.
Total Number of Observations Read from WORK.COMPARE2: 3.

Number of Observations with Some Compared Variables Unequal: 2.
Number of Observations with All Compared Variables Equal: 1.
```

```
                    Values Comparison Summary
Number of Variables Compared with All Observations Equal: 1.
Number of Variables Compared with Some Observations Unequal: 2.
Total Number of Values which Compare Unequal: 2.
Maximum Difference: 0.1.

                 Variables with Unequal Values
Variable      Type   Len    Label                Ndif   MaxDif
studentID     CHAR     8                           1
weight1       NUM      8    first weight value     1    0.100

              Value Comparison Results for Variables

                                 ||     Base Value    Compare Value
      Obs                        ||     studentID     studentID
 _______                         ||     _________     _________
                                 ||
       3                         ||     A03           A04
```

```
                                 ||     first weight value
                                 ||        Base       Compare
      Obs                        ||     weight1       weight1      Diff.    % Diff
 _______                         ||     _______     _________    _______  _______
                                 ||
       2                         ||      150.20      150.3000     0.1000   0.0666
```

10.2.2 Comparing Observations with Common ID Values

In Program 10.3, values are compared based on the observations that occur in the same position. In this situation, you need to ensure the matching observations are from the same person or from the same sources. For example, the third observation in COMPARE1 has a value of "A03" for ID, whereas ID equals "A04" in the third observation of COMPARE2.

The results from the comparison in Program 10.3 might be just what you are looking for. In this situation, you can use the ID statement to confine your comparisons to those observations where the *ID* variable values match. To include the ID statement in PROC COMPARE, input data sets must be previously sorted by the named ID variable.

Program 10.4 uses STUDENTID in the ID statement to compare observations that have matching values for STUDENTID. Because the input data set has been created in STUDENTID order, there is no need to sort by STUDENTID again in the program.

Because the BY statement produces separate comparisons for male and female students, both data sets require a previous sort by the same BY variable. The VAR statement is used to restrict the comparison to the WEIGHT1 variable in both data sets.

By default, PROC COMPARE generates lengthy reports of comparison between the base and comparison data sets. The BRIEFSUMMARY (or BRIEF) option produces a short comparison between base and comparison data sets.

Program 10.4:

```
proc sort data = compare1;
    by gender;
run;

proc sort data = compare2;
    by gender;
run;

title 'PROC COMPARE: ID, BY and VAR statement';
proc compare base = compare1
             compare = compare2
             brief;
    by gender;
    id studentID;
    var weight1;
run;
```

Output from Program 10.4:

```
            PROC COMPARE: ID, BY and VAR statement
                    The COMPARE Procedure
         Comparison of WORK.COMPARE1 with WORK.COMPARE2
                      (Method = EXACT)

- - - - - - - - - - - - - - - - - - - gender = F- - - - - - - - - - - - - - -
NOTE: Data set WORK.COMPARE1 contains 1 observations not in
      WORK.COMPARE2.
NOTE: Values of the following 1 variables compare unequal:
      weight1

             Value Comparison Results for Variables

                ||  first weight value
                ||     Base     Compare
   studentID    ||  weight1     weight1        Diff.        % Diff
   _________    ||  _______    _________    _________     ________
                ||
   A04          ||   149.20    149.1000      -0.1000       -0.0670

- - - - - - - - - - - - - - - - - - - gender = M- - - - - - - - - - - - - - -
NOTE: Values of the following 1 variables compare unequal:
      weight1
```

```
            Value Comparison Results for Variables

                ||  first weight value
                ||     Base     Compare
   studentID    ||  weight1     weight1       Diff.       % Diff
   _________    ||  _______     _______     ________     ________
                ||
   A02          ||   150.20    150.3000      0.1000       0.0666
```

In the last two examples from this section, variables with the same name have been compared. However, it is possible to compare variables with different names.

Program 10.5 compares WEIGHT2 that is specified in the VAR statement from the *base* data set with WEIGHT1 specified by the WITH statement from the *comparison* data set. The ID statement also guarantees that the comparisons will be made between observations with matching values for STUDENTID.

Program 10.5:

```
title 'Compare variables with different variable names in
different data sets';
proc compare base = compare1
             compare = compare2
             brief;
    id studentID;
    var weight2;
    with weight1;
run;
```

Output from Program 10.5:

```
Compare variables with different variable names in different
data sets
                    The COMPARE Procedure
        Comparison of WORK.COMPARE1 with WORK.COMPARE2
                        (Method = EXACT)
NOTE: Data set WORK.COMPARE1 contains 1 observations not in
      WORK.COMPARE2.
NOTE: Values of the following 1 variables compare unequal:
      weight2^ = weight1

            Value Comparison Results for Variables

                ||     Base     Compare
   studentID    ||  weight2     weight1       Diff.       % Diff
   _________    ||  _______     _______     ________     ________
                ||
   A04          || 149.2000   149.1000      -0.1000      -0.0670
```

```
    A01       || 154.7000   154.3000      -0.4000       -0.2586
    A02       || 150.2000   150.3000       0.1000        0.0666
```

You do not need the COMPARE= option to compare variables within the same data set. Program 10.6 compares two variables, WEIGHT1 and WEIGHT2, within the COMPARE1 data set.

Program 10.6:

```
title 'Compare variables with different variable names in the
same data sets';
proc compare base = compare1 brief;
    var weight1;
    with weight2;
run;
```

Output from Program 10.6:

```
Compare variables with different variable names in the
same data sets
                    The COMPARE Procedure
           Comparisons of variables in WORK.COMPARE1
                        (Method = EXACT)
NOTE: Values of the following 1 variables compare unequal:
      weight1^ = weight2

             Value Comparison Results for Variables

      || first weight value
      ||       Base        Compare
  Obs ||     weight1       weight2        Diff.        % Diff
 ____ ||     _______      ________     _______       _______
      ||
   1  ||      149.10       149.2000      0.1000        0.0671
   3  ||      154.30       154.7000      0.4000        0.2592
```

10.3 Restructuring Data Sets Using the TRANSPOSE Procedure

Restructuring or transposing data sets were first introduced in Chapter 3 and Chapter 4 by using the DATA step programming. Chapter 6 introduced how to use array processing to efficiently transpose data. Another method for restructuring a data set is to use the TRANSPOSE procedure.

The material in this section is based upon a paper that I presented at SAS Global Forum (Li, 2012).

The syntax of PROC TRANSPOSE is listed below. The six statements in the TRANSPOSE procedure, which includes the PROC TRANPOSE, BY, COPY, ID, IDLABEL, and VAR statements, along with the eight options in the PROC TRANSPOSE statement, are used to apply different types of data transpositions and give the resulting data set a different appearance.

```
PROC TRANSPOSE <DATA=input-data-set>
               <DELIMITER=delimiter>
               <LABEL=label>
               <LET>
               <NAME=name>
               <OUT=output-data-set>
               <PREFIX=prefix>
               <SUFFIX=suffix>;
   BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>;
   COPY variable(s);
   ID variable;
   IDLABEL variable;
   VAR variable(s);
RUN;
```

10.3.1 Transposing an Entire Data Set

Program 10.7 starts by creating data set DAT1 with variables S_NAME, S_ID, E1, E2, and E3. The three numeric variables, E1 through E3, are labeled "English1", "English2", and "English3". By default, without specifying the names of the transposing variables, all the numeric variables from the input data set are automatically transposed.

In Program 10.7, the three numeric variables containing English test scores for the two students structured as a 2-by-3 matrix become three *observations* in a 3-by-2 matrix that defines the transposed data set for the two students newly represented as *variables*.

In the PROC TRANSPOSE statement, the OUT= option is used to specify the name of the transposed data set. Without an OUT= option, PROC TRANSPOSE will create a data set that uses the DATA*n* naming convention.

The NAME= option references the variable in the transposed data set that contains the names of the variables from the original data set which are being transposed. Without the NAME= option, the default _NAME_ is used.

Because E1 through E3 variables have permanent labels from the input data set, these labels are stored under the variable specified in the LABEL= option. Without specifying the LABEL = option, the default name _LABEL_ is used.

The PREFIX= option adds a prefix to the transposed variable names. Given PREFIX= score_, for example, the names of the transposed variables change to SCORE_1 and SCORE_2. Without the PREFIX= option, SAS uses the defaults, COL1 and COL2. You can also use the SUFFIX= option to attach a suffix to the transposed variable name.

Program 10.7:

```
data dat1;
    input s_name $ s_id $ e1 - e3;
    label e1 = English1
          e2 = English2
          e3 = English3;
    datalines;
John A01 89 90 92
Mary A02 92 .  81
;

proc transpose data = dat1
               out = dat1_out1
               name = varname
               label = labelname
               prefix = score_;
run;

title 'Dat1 in the original form';
proc print data = dat1 label;
run;

title 'Dat1 in transposed form';
proc print data = dat1_out1;
run;
```

Output from Program 10.7:

Dat1 in the original form

Obs	s_name	s_id	English1	English2	English3
1	John	A01	89	90	92
2	Mary	A02	92	.	81

Dat1 in transposed form

Obs	varname	labelname	score_1	score_2
1	e1	English1	89	92
2	e2	English2	90	.
3	e3	English3	92	81

Program 10.8 adds more statements to PROC TRANSPOSE. The VAR statement is used to specify the variables that are to be transposed. In this instance, the VAR statement references the same E1 through E3 variables that are transposed by default. Thus, the output from Program 10.7 and Program 10.8 is the same.

The ID statement is used to specify the variable from the input data set that contains the values to rename the transposed variables. Because two variables S_NAME and S_ID are used in the ID statement along with the DELIM= option in the PROC TRANSPOSE statement, the values created by concatenating the S_NAME and the S_ID variables (separated by the value specified by the DELIM= option) are used as the names of the transposed variables.

The variable specified in the IDLABEL statement from the input data set can be either numeric or character and contains the values to label the transposed variables in the newly created transposed data set. From the partial output from the CONTENTS procedure, you can see that the names of the transposed variables are JOHN_A01 and MARY_A02, with A01 and A02 as their labels, respectively.

Program 10.8:

```
proc transpose data = dat1
               out = dat1_out2
               label = labelname
               name = varname
               delim = _;
    var e1-e3;
    id s_name s_id;
    idlabel s_id;
run;

title 'The use of VAR, ID, and IDLABEL statements';
proc print data = dat1_out2;
run;

proc contents data = dat1_out2;
run;
```

Output from Program 10.8:

```
      The use of VAR, ID, and IDLABEL statements
   Obs    varname    labelname    John_A01    Mary_A02
    1       e1        English1        89          92
    2       e2        English2        90          .
    3       e3        English3        92          81
```

Partial Output from PROC CONTENTS from Program 10.8:

```
      Alphabetic List of Variables and Attributes
   #    Variable     Type    Len    Label
   3    John_A01     Num      8     A01
   4    Mary_A02     Num      8     A02
   2    labelname    Char    40     LABEL OF FORMER VARIABLE
   1    varname      Char     8     NAME OF FORMER VARIABLE
```

10.3.2 Introduction to Transposing BY Groups

You can also transpose the data set BY group. More than one variable can be listed in the BY statement. To use the BY statement in PROC TRANSPOSE, the data set must be previously sorted using the same BY variable. When transposing a data set BY group, the BY variable is not transposed.

Program 10.9 transposes DAT1 by using S_NAME as the BY variable. The number of observations in the transposed data set (6) equals the number of BY groups (2) times the number of variables that are transposed (3). The number of transposed variables is equal to the number of observations within each BY group in the sorted input data set. For the input data processed in Program 10.9, the number of observations within each BY group is just one. Therefore, the number of transposed variables in the output data set is also just one (COL1).

The COPY statement in Program 10.9 copies the values from the S_ID variable from the input data set directly to the transposed data set. Because there are two observations from the input data set, the number of observations that will be copied will be two as well; SAS pads the missing values to the rest of the observations.

Program 10.9:

```
proc sort data = dat1 out = dat1_sort;
    by s_name;
run;

title 'DAT1_SORT where DAT1 is Sorted by S_NAME';
proc print data = dat1_Sort;
run;

proc transpose data = dat1_sort
               out = dat1_out3
               name = varname
               label = TEST;
    by s_name;
    copy s_id;
run;

title 'Transposing BY-group with the COPY statement';
proc print data = dat1_out3;
run;
```

Output from Program 10.9:

```
                DAT1_SORT where DAT1 is Sorted by S_NAME
          Obs   s_name   s_id    e1     e2    e3
           1     John     A01    89     90    92
           2     Mary     A02    92      .    81
```

```
  Transposing BY-group with the COPY statement
Obs   s_name  s_id   varname      TEST     COL1
 1     John    A01      e1       English1   89
 2     John             e2       English2   90
 3     John             e3       English3   92
 4     Mary    A02      e1       English1   92
 5     Mary             e2       English2    .
 6     Mary             e3       English3   81
```

10.3.3 Where the ID Statement Does Not Work for Transposing BY Groups

You can use the ID statement to specify the variable from the input data set that contains the values to use for renaming transposed variables. In Program 10.9, the newly created transposed variable is given the uninformative name of COL1 by default. However, if you want to use the S_ID variable in the ID statement to change COL1, the results from Program 10.10 won't conform to your expectations. Now the transposed values in the output occupy *two* columns with A01 and A02 as variable names. What you expected is output where COL1 is simply renamed. To rename COL1, remove the ID statement and issue a RENAME suboption in the OUT= option attached to the PROC TRANSPOSE statement.

Program 10.10:

```
proc transpose data = dat1_sort
               out = dat1_out4
               name = varname
               label = TEST;
    by s_name;
    id s_id;
run;

title 'Incorrect way to use the ID statement';
proc print data = dat1_out4;
run;

proc transpose data = dat1_sort
               out = dat1_out5(rename = (col1 = Score))
               name = varname
               label = TEST;
    by s_name;
run;

title 'Rename COL1 to SCORE in the Output Data Set';
proc print data = dat1_out5;
run;
```

Output from Program 10.10:

```
Incorrect way to use the ID statement
Obs   s_name   varname     TEST      A01   A02
 1     John       e1      English1   89     .
 2     John       e2      English2   90     .
 3     John       e3      English3   92     .
 4     Mary       e1      English1    .     92
 5     Mary       e2      English2    .      .
 6     Mary       e3      English3    .     81

Rename COL1 to SCORE in the Output Data Set
Obs   s_name   varname     TEST      Score
 1     John       e1      English1    89
 2     John       e2      English2    90
 3     John       e3      English3    92
 4     Mary       e1      English1    92
 5     Mary       e2      English2     .
 6     Mary       e3      English3    81
```

10.3.4 Where the ID Statement Is Essential for Transposing BY Groups

Program 10.11 illustrates a situation where the ID statement is necessary in order to transpose data correctly. PROC TRANSPOSE in program 10.11 transposes one variable, SCORE, by using the variable S_NAME as the BY variable. The resulting transposed data set has two observations, which equals the number of BY groups (2) times the number of variables that are transposed (1). The problem with the transposed data set is that the third test score (81) for Mary is placed in the location reserved for the second test score.

Program 10.11:

```
data dat2;
    input s_name $ s_id $ exam score;
    datalines;
John A01 1 89
John A01 2 90
John A01 3 92
Mary A02 1 92
Mary A02 3 81
;

proc sort data = dat2 out = dat2_sort;
    by s_name;
run;

proc transpose data = dat2_sort out = dat2_out1;
    var score;
    by s_name;
run;
```

```
title 'Incorrect way to transpose - ID statement is not used';
proc print data = dat2_out1;
run;
```

Output from Program 10.11:

```
Incorrect way to transpose - ID statement is not used
 Obs   s_name   _NAME_  COL1  COL2  COL3
  1     John    score    89    90    92
  2     Mary    score    92    81     .
```

Program 10.12 fixes the problem in Program 10.11 by using an ID statement along with the PREFIX = option. Both have to be used because ID references numeric variable EXAM. Without a user-defined PREFIX, the procedure would automatically insert an underscore before the numeric values in EXAM to create valid SAS variable names. Instead of the desired "test_1", you would get an inscrutable "_1".

Program 10.12:

```
proc transpose data = dat2_sort
               out = dat2_out2(drop = _name_)
               prefix = test_;
    var score;
    by s_name;
    id exam;
run;

title 'Correct way to transpose - ID statement is used';
proc print data = dat2_out2;
run;
```

Output from Program 10.12:

```
Correct way to transpose - ID statement is used
   Obs    s_name    test_1    test_2    test_3
    1      John       89        90        92
    2      Mary       92         .        81
```

10.3.5 Handling Duplicated Observations Using the LET Option

John and Mary have two different scores for their third test in the DAT3 data set created in Program 10.13. If you use S_NAME as the BY variable and specify EXAM in the ID statement, PROC TRANSPOSE will not be able to transpose DAT3 and will generate the following error message written twice to the log:

```
ERROR: The ID value "test_3" occurs twice in the same BY group.
```

If the ID statement is removed from the code, PROC TRANSPOSE will be able to transpose DAT3, but the results will be sent to four columns, not three, because the maximum number of observations per BY group is four with the double entry for John.

For situations with duplicated records, you may want to keep only one record, such as keeping the largest or the smallest of the duplicated entries. The LET option from the PROC TRANSPOSE statement allows you to keep the last occurrence of a particular ID value within either the entire data set or a BY group.

Program 10.13 transposes DAT3 by keeping the largest value of each EXAM within each group of the S_NAME variable. Thus, it is necessary to sort the data by S_NAME first, followed by EXAM, and then finally by SCORE, in ascending order. Because of the sort, the highest score ends up being the last observation for each EXAM a student takes. Duplicate entries are not eliminated from the input data. Instead, SAS writes the same error message to the log; this time, however, the error is upgraded to a warning, so the data are successfully and accurately transposed.

Program 10.13:

```
data dat3;
    input s_name $ s_id $ exam score;
    datalines;
John A01 1 89
John A01 2 90
John A01 3 92
John A01 3 95
Mary A02 1 92
Mary A02 3 81
Mary A02 3 85
;

proc sort data = dat3 out = dat3_sort;
    by s_name exam score;
run;

proc transpose data = dat3_sort
               out = dat3_out(drop = _name_)
               prefix = test_
               let;
    var score;
    by s_name;
    id exam;
run;

title 'Keep the maximum score';
proc print data = dat3_out;
run;
```

Output from Program 10.13:

```
            Keep the maximum score
       Obs  s_name  test_1  test_2  test_3
        1    John      89      90      95
        2    Mary      92       .      85
```

Partial Log from Program 10.13:

```
69     proc transpose data = dat3_sort
70                    out = dat3_out(drop = _name_)
71                    prefix = test_
72                    let;
73          var score;
74          by s_name;
75          id exam;
76     run;

WARNING: The ID value "test_3" occurs twice in the same BY
         group.
NOTE: The above message was for the following BY group:
      s_name = John
WARNING: The ID value "test_3" occurs twice in the same BY
         group.
NOTE: The above message was for the following BY group:
      s_name = Mary
NOTE: There were 7 observations read from the data set
      WORK.DAT3_SORT.
NOTE: The data set WORK.DAT3_OUT has 2 observations and 4
      variables.
NOTE: PROCEDURE TRANSPOSE used (Total process time):
      real time     0.00 seconds
      cpu time      0.00 seconds
```

If you want to keep the smallest SCORE instead of the largest in the transposed data, all you need to do is sort S_NAME and EXAM in ascending order and then sort SCORE in descending order.

10.3.6 Situations Requiring Two or More Transpositions

In some applications, a single transposition will not produce the desired results. For example, to transpose DAT4 to DAT4_TRANSPOSE, you need to use PROC TRANSPOSE twice. Program 10.14A starts by creating the data set DAT4 and follows by transposing DAT4 by variable S_NAME.

DAT4:

	S_NAME	E1	E2	E3	M1	M2	M3
1	John	89	90	92	78	89	90
2	Mary	92	.	81	76	91	89

DAT4_TRANSPOSE:

	TEST_NUM	JOHN_E	JOHN_M	MARY_E	MARY_M
1	1	89	78	92	76
2	2	90	89	.	91
3	3	92	90	81	89

Program 10.14A:

```
data dat4;
   input s_name $ E1 - E3 M1 - M3;
datalines;
John 89 90 92 78 89 90
Mary 92 .  81 76 91 89
;

proc sort data = dat4 out = dat4_sort1;
    by s_name;
run;

proc transpose data = dat4_sort1 out = dat4_out1;
    by s_name;
run;

title 'First use of PROC TRANSPOSE for dat4';
proc print data = dat4_out1;
run;
```

Output from Program 10.14A:

```
First use of PROC TRANSPOSE for dat4
     Obs    s_name   _NAME_   COL1
      1      John       E1      89
      2      John       E2      90
      3      John       E3      92
      4      John       M1      78
      5      John       M2      89
      6      John       M3      90
      7      Mary       E1      92
      8      Mary       E2      .
      9      Mary       E3      81
     10      Mary       M1      76
     11      Mary       M2      91
     12      Mary       M3      89
```

Before performing a second transpose, you need to preprocess the output data set DAT4_OUT1 created in the first transposition. More specifically, two new variables have to be created: CLASS and TEST_NUM. CLASS is the first character in _NAME_, and TEST_NUM is the second and final character in _NAME_. Program 10.14B uses the SUBSTR function to create both TEST_NUM and CLASS.

Program 10.14B:

```
data dat4_out1a;
    set dat4_out1;
    test_num = substr(_name_,2);
    class = substr(_name_,1,1);
run;

title 'Creating TEST_NUM and CLASS variables';
proc print data = dat4_out1a;
run;
```

Output from Program 10.14B:

Creating TEST_NUM and CLASS variables

Obs	s_name	_NAME_	COL1	test_num	class
1	John	E1	89	1	E
2	John	E2	90	2	E
3	John	E3	92	3	E
4	John	M1	78	1	M
5	John	M2	89	2	M
6	John	M3	90	3	M
7	Mary	E1	92	1	E
8	Mary	E2	.	2	E
9	Mary	E3	81	3	E
10	Mary	M1	76	1	M
11	Mary	M2	91	2	M
12	Mary	M3	89	3	M

Program 10.14C sorts the data by TEST_NUM, within TEST_NUM by S_NAME, and finally within S_NAME by CLASS. Notice that the test scores in COL1 now have the desired order for the final transposition.

Program 10.14C:

```
proc sort data = dat4_out1a out = dat4_sort2;
    by test_num s_name class;
run;

title 'Sort data by TEST_NUM, S_NAME and CLASS';
proc print data = dat4_sort2;
run;
```

Output from Program 10.14C:

Sort data by TEST_NUM, S_NAME and CLASS

Obs	s_name	_NAME_	COL1	test_num	class
1	John	E1	89	1	E
2	John	M1	78	1	M
3	Mary	E1	92	1	E

```
4       Mary      M1      76        1        M
5       John      E2      90        2        E
6       John      M2      89        2        M
7       Mary      E2      .         2        E
8       Mary      M2      91        2        M
9       John      E3      92        3        E
10      John      M3      90        3        M
11      Mary      E3      81        3        E
12      Mary      M3      89        3        M
```

PROC TRANSPOSE in Program 10.14D transposes COL1 by variable TEST_NUM and uses S_NAME and CLASS as the ID variables. The names of the transposed variables are separated by the underscore from the DELIMITER= option.

Program 10.14D:

```
proc transpose data = dat4_sort2
               out = dat4_out2(drop = _name_)
               delimiter = _;
    by test_num;
    var col1;
    id s_name class;
run;

title 'Second use of PROC TRANSPOSE for dat4';
proc print data = dat4_out2;
run;
```

Output from Program 10.14D:

```
        Second use of PROC TRANSPOSE for dat4
 Obs    test_num    John_E    John_M    Mary_E    Mary_M
  1         1          89        78        92        76
  2         2          90        89         .        91
  3         3          92        90        81        89
```

10.4 Creating the User-Defined Format Using the FORMAT Procedure

SAS provides a large selection of ready-made formats described earlier in Chapters 1 and 8. If you cannot find a format provided by SAS, you can create your own with the FORMAT procedure. Besides customized formats, PROC FORMAT can be used to create informats, list the contents of format catalogs, generate formats and informats from SAS data sets, and conversely create SAS data sets from formats or informats.

This section focuses on creating and printing customized formats and utilizing formats to create variables. Only the PROC FORMAT and the VALUE statements, listed in the syntax below, are presented in this section. Readers should refer to SAS documentation for other features of PROC FORMAT.

```
PROC FORMAT <LIBRARY=libref<.catalog>> < FMTLIB>;
   VALUE <$>name value-range-set(s);
RUN;
```


10.4.1 Creating User-Defined Formats

PROC FORMAT stores user-defined formats as entries in SAS catalogs. Without specifying the LIBRARY=*libref* option in the PROC FORMAT statement, newly created formats will be stored in the WORK.FORMATS catalog. WORK.FORMATS exists only for the duration of the current SAS session. If you specify *libref* without a *catalog* name, your formats will be stored in the *libref*.FORMATS catalog. For example, specifying the following statements, PROC FORMAT stores defined formats in the A.FORMATS catalog located in 'C:\" as FORMATS.SAS7BCAT.

```
libname a 'C:\';
proc format library = a;
```

Specifying LIBRARY=*libref.catalog-name*, the defined formats will be stored in the *libref.catalog-name* catalog. For example, specifying the following statements, PROC FORMAT stores defined formats in the A.MYFORMAT catalog located in 'C:\" as MYFORMAT.SAS7BCAT.

```
libname a 'C:\';
proc format library = a.myformat;
```

The VALUE statement is used to create a format that specifies character strings to use to print variable values. The *name* component in the VALUE statement is the name of the format that you are creating. Your format name cannot be the same as the name of a SAS-supplied format and must be a valid SAS name. Format names can have up to 32 characters. If a character format is being defined, the dollar sign ($) that appears in the first position counts as one of the 32 characters. Furthermore, a format name cannot end with a number because trailing numbers indicate lengths when formats are actually being applied. When creating a format in the VALUE statement, do not use a period after the format name.

The *value-range-set(s)* in the VALUE statement is used to specify one or more variable values and a character string. You can specify one or more *value-range-set(s)* in the following form:

```
value-or-range-1 <..., value-or-range-n>='formatted-value'
```

The *'formatted-value'* on the right side of the equal sign is used to specify a character string that becomes the printed value of the corresponding variable values listed as *value-or-range* sets that appears on the left side of the equal sign. The *'formatted-value'* must be character strings and can be up to 32,767 characters.

Based on the syntax above, you can specify multiple occurrences of *value-or-range*, separated by commas, in a value-range-set on the left side of the equal sign. Each occurrence of *value-or-range* can be specified as either *value* or *range*, which follows certain rules. Examples of specifying *value-or-range* are provided in Table 10.1.

The *value* in an occurrence of *value-or-range* is a single value, for example, 'A' or 1. For character formats, you need to enclose the character values, up to 32,767 characters, in single quotation marks. The *range* is used to specify a list of values, such as 5-10 (values from 5 to 10), or 'A'-'D' (values from 'A' to 'D').

The "less than" (<) symbol can be used to exclude values from ranges. If you are excluding the first value in a range, then you need to put the less than symbol (<) after the value. Similarly, if you are excluding the last value in a range, then put the < before the values. (See Example 3 in Table 10.1.)

If a value appears at the high end of one range as well as at the low end of another range, and the less than (<) symbol is not used, then PROC FORMAT assigns the value to the first range. (See Example 4 in Table 10.1.) If you overlap values in ranges, PROC FORMAT will issue an error message unless the MULTILABEL option is used (not covered in this book).

You can use the keywords LOW and HIGH as values that represent the lowest and highest numeric or character values in a range. The keyword LOW does not include numeric missing values, but it does include character missing values. You can also use the keyword OTHER to match all values that do not match any value or range or missing values. If LOW is used in a character range and OTHER is used as well, the missing value will not be matched in the OTHER group. (See Example 7 in Table 10.1.) Other examples of labeling missing character and numeric values and how the missing values are matched when the keywords OTHER and LOW are used are listed in Table 10.1.

Program 10.15 uses three FORMAT procedures and stores the $RACE, YESNO, $SMKFMT, SALFMT, and AGEFMT formats in three different catalogs. The first FORMAT procedure stores the $RACE YESNO and $SMKFMT formats in the FORMATS catalog in the WORK library. The $SMKFMT format is used for formatting values 'past' and 'never' with '0' and 'current' with '1'. Even though 0 and 1 are numeric values, they still need to be placed in quotation marks. The second FORMAT procedure stores the SALFMT format in the SALARY catalog in the DESKTOP library. The third FORMAT procedure stores the AGEFMT format in the FORMATS catalog in the DESKTOP library.

TABLE 10.1

Examples of Value-or-Range Sets

Example	All Possible Values	*Value-or-Range*	Values Being Labeled
1	2, 3, 4, 5, 6, 7, 8, and 9	`3, 5, 7, 9 = 'odd'` `2, 4, 6, 8 = 'even'`	3, 5, 7, and 9 are labeled with 'odd' 2, 4, 6, and 8 are labeled with 'even'
2	'A', 'B', 'C', 'D', and 'E'	`'A', 'C', 'E' = 'grp1'` `'B', 'D' = 'grp2'`	'A', 'C', and 'E' are labeled with 'grp1' 'B' and 'D' are labeled with 'grp2'
3	2, 3, 4, 5, 6, 7, 8, and 9	`2<-<6 = 'low'` `6-<9 = 'high'`	3, 4, and 5 are labeled with 'low' 6, 7, and 8 are labeled with 'high' 2 and 9 will be printed as 2 and 9 without any labels
4	2, 3, 4, 5, 6, 7, 8, and 9	`2-6 = 'low'` `6-9 = 'high'`	2, 3, 4, 5, and 6 are labeled with 'low' 7, 8, and 9 are labeled with 'high'

5	2, 3, 4, 5, 6, 7, 8, 9, and missing (.)	`LOW-6 = 'low'`
		`7-HIGH = 'high'`
		`OTHER = 'missing'`
			2, 3, 4, 5, and 6 are labeled with 'low'
			7, 8, and 9 are labeled with 'high'
			The missing value (.) is labeled with 'missing'

6	2, 3, 4, 5, 6, 7, 8, 9, and missing (.)	`LOW-6 = 'low'`
		`7-HIGH = 'high'`
		`. = 'missing'`
			2, 3, 4, 5, and 6 are labeled with 'low'
			7, 8, and 9 are labeled with 'high'
			The missing value (.) is labeled with 'missing'

7	'A', 'B', 'C', 'D', 'E', and missing (' ')	`LOW-'C' = 'low'`
		`'D'-HIGH = 'high'`
		`OTHER = 'missing'`
			'A', 'B', 'C', and missing value are labeled with 'low'
			'D' and 'E' are labeled with 'high'

8	'A', 'B', 'C', 'D', 'E', and missing (' ')	`'A'-'C' = 'low'`
		`'D'-HIGH = 'high'`
		`OTHER = 'missing'`
			'A', 'B', and 'C' are labeled with 'low'
			'D' and 'E' are labeled with 'high'
			The missing value is labeled with 'missing'

9	'A', 'B', 'C', 'D', 'E', and missing (' ')	`'A'-'C' = 'low'`
		`'D'-HIGH = 'high'`
		`' ' = 'missing'`
			'A', 'B', and 'C' are labeled with 'low'
			'D' and 'E' are labeled with 'high'
			The missing value is labeled with 'missing'

Program 10.15:

```
libname desktop 'C:\Documents and Settings\All Users\Desktop';
libname library 'C:\Documents and Settings\ARTHUR\Desktop\
   saslib';
proc format;
    value $race
        'A' = 'Asian'
        'B' = 'African American'
        'H' = 'Hispanic'
        'W' = 'White';
    value yesno
        1 = 'Yes'
        0 = 'No';
    value $smkfmt 'past', 'never' = '0'
          'current' = '1';
run;

proc format library = desktop.salary;
    value salfmt
        low - <50000 = 'low'
        50000 - <100000 = 'average'
        100000 - high = 'high';
run;

proc format library = desktop;
    value agefmt
        low - <30 = '< 30'
        30 - high = '> = 30'
        other = 'missing';
run;
```

Once the formats are created, you can use the FMTLIB option in the PROC FORMAT statement to display the contents of the format catalog. Program 10.16 prints the contents of the formats from the WORK.FORMAT, DESKTOP.FORMAT and DESKTOP.SALARY catalogs.

Program 10.16:

```
title 'Contents in WORK.FORMAT catalog';
proc format fmtlib;
run;

title 'Contents in DESKTOP.FORMAT catalog';
proc format library = desktop fmtlib;
run;

title 'Contents in DESKTOP.SALARY catalog';
proc format library = desktop.salary fmtlib;
run;
```

Output from Program 10.16:

Contents in WORK.FORMAT catalog

FORMAT NAME: YESNO LENGTH: 3 NUMBER OF VALUES: 2		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 3 FUZZ: STD		
START	END	LABEL (VER. V7|V8 14AUG2012:11:06:48)
0	0	No
1	1	Yes

FORMAT NAME: $RACE LENGTH: 16 NUMBER OF VALUES: 4		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 16 FUZZ: 0		
START	END	LABEL (VER. V7|V8 14AUG2012:11:06:48)
A	A	Asian
B	B	African American
H	H	Hispanic
W	W	White

FORMAT NAME: $SMKFMT LENGTH: 1 NUMBER OF VALUES: 3		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 1 FUZZ: 0		
START	END	LABEL (VER. V7|V8 14AUG2012:11:06:48)
current	current	1
never	never	0
past	past	0

Contents in DESKTOP.FORMAT catalog

FORMAT NAME: AGEFMT LENGTH: 7 NUMBER OF VALUES: 3		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 7 FUZZ: STD		
START	END	LABEL (VER. V7|V8 14AUG2012:09:26:30)
LOW	30<< 30	
30	HIGH	> = 30
OTHER	**OTHER**	missing

```
                Contents in DESKTOP.SALARY catalog
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
| FORMAT NAME: SALFMT LENGTH: 7 NUMBER OF VALUES: 3                     |
| MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH 7 FUZZ: STD               |
+- - - - - - +- - - - - +- - - - - - - - - - - - - - - - - - - - - - - +
|START        |END       |LABEL (VER. V7|V8 14AUG2012:09:26:29)|
+- - - - - - +- - - - - +- - - - - - - - - - - - - - - - - - - - - - - +
|LOW          |     50000<low                                         |
|        50000|    100000<average                                     |
|       100000|HIGH      |high                                        |
+- - - - - - +- - - - - +- - - - - - - - - - - - - - - - - - - - - - - +
```

10.4.2 Retrieving User-Defined Formats

Once formats are defined, you can associate them with the variables either permanently in the DATA step or temporarily in a PROC step. To retrieve a temporary format that is stored in the WORK library, you need to include the name of the format in the FORMAT statement because SAS automatically looks for the format in the WORK.FORMATS catalog. In addition, SAS automatically searches for formats in the LIBRARY.FORMATS. However, if LIBRARY is not used as the *libref* name or FORMATS is not specified as the catalog name, you must use the FMTSEARCH= system option and include the *libref* name or the catalog name or both in an OPTIONS statement; otherwise, SAS will not be able to locate your formats. Here is the syntax for specifying a list of format catalogs to search in the OPTIONS statement:

```
OPTIONS FMTSEARCH=(catalog-specification-1... catalog-
   specification-n);
```

Each *catalog-specification* can be *libref* or *libref.catalog*. If only *libref* is specified, then SAS assumes the catalog name is FORMATS. SAS searches the WORK. FORMATS catalog first, and then the LIBRARY.FORMATS catalog; SAS then searches the catalogs in the FMTSEARCH = list in the order in which they are listed until the format is found.

The HEARING data set used in Program 10.17 was created in Chapter 1. Additional information about the data set can be found in Table 1.1. Program 10.17 associates five user-defined formats, $RACE, YESNO, $SMKFMT, AGEFMT, and SALFMT, permanently to the variables RACE, PREG, SMOKE, AGE, and INCOME, respectively. Access to all formats is made possible by the FMTSEARCH option.

The two PRINT procedures in Program 10.17 show the impact that format assignments have upon printed output. PROC FREQ demonstrates how cross-tabulation of the two continuous variables AGE and INCOME is enhanced by using their formatted values.

Program 10.17:

```
options fmtsearch = (desktop desktop.salary);
data hearing2;
    set hearing;
    format race $race.
           preg yesno.
           smoke $smkfmt.
           age agefmt.
           income salfmt.;
run;

title 'Data set HEARING, formats are not used';
proc print data = hearing (obs = 5);
    var id race preg smoke age income;
run;

title 'Data set HEARING2, formats are used';
proc print data = hearing2 (obs = 5);
    var id race preg smoke age income;
run;

title 'Cross tabulation of AGE and INCOME by using formatted
value';
proc freq data = hearing2;
    tables age*income/missing;
run;
```

Output from Program 10.17:

```
                Data set HEARING, formats are not used
      Obs    id      race    Preg    smoke    Age    Income
       1    629F      H       0     past      26     35000
       2    656F      W       1     never     26     48000
       3    711F      W       1     never     32     30000
       4    511F      B       0     never     32     25000
       5    478F      W       0     past      34     35700

                  Data set HEARING2, formats are used
    Obs    id      race              Preg   smoke    Age     Income
     1    629F    Hispanic           No       0     < 30      low
     2    656F    White              Yes      0     < 30      low
     3    711F    White              Yes      0     > = 30    low
     4    511F    African American   No       0     > = 30    low
     5    478F    White              No       0     > = 30    low

Cross tabulation of AGE and INCOME by using formatted value
                        The FREQ Procedure
                      Table of Age by Income
```

Age Frequency Percent Row Pct Col Pct	Income low	average	high	Total
missing	0	0	1	1
	0.00	0.00	2.94	2.94
	0.00	0.00	100.00	
	0.00	0.00	33.33	
< 30	14	6	1	21
	41.18	17.65	2.94	61.76
	66.67	28.57	4.76	
	70.00	54.55	33.33	
> = 30	6	5	1	12
	17.65	14.71	2.94	35.29
	50.00	41.67	8.33	
	30.00	45.45	33.33	
Total	20	11	3	34
	58.82	32.35	8.82	100.00

10.4.3 Creating Variables by Using User-Defined Formats

You can create categorical or indicator variables in a DATA step by exercising user-defined formats with PUT and INPUT functions. For example, Program 10.18 creates two variables, PREG_CHAR and SMOKE_IND, by using two of the formats created in Program 10.15.

Recall from Chapter 9 that the PUT function can convert either numeric values with numeric formats or character values with character formats to character values. Thus, the results generated from the PUT function are always characters. Because the PREG variable is numeric, the numeric format (YESNO) is used to convert PREG to the character variable PREG_CHAR.

To create the indicator variable SMOKE_IND, the PUT function converts the values from the SMOKE variable by using the $SMKFMT format to character values '1' and '0' first. Then the INPUT function converts the character results generated from the PUT function to numeric values by using the *w.* informat.

Program 10.18:

```
data hearing3;
    set hearing;
    preg_char = put(preg, yesno.);
    smoke_ind = input(put(smoke, $smkfmt.), 1.);
run;
```

```
title 'Checking if PREG_CHAR and SMOKE_IND are created
correctly';
proc freq data = hearing3;
    tables preg*preg_char
           smoke*smoke_ind/nocol norow nopercent missing;
run;
```

Output from Program 10.18:

```
Checking if PREG_CHAR and SMOKE_IND are created correctly
                    The FREQ Procedure
                Table of Preg by preg_char
       Preg       preg_char
       Frequency|       . |No       |Yes      | Total
       - - - - -+- - - - +- - - - +- - - - +
              . |       4 |       0 |       0 |     4
       - - - - -+- - - - +- - - - +- - - - +
              0 |       0 |      19 |       0 |    19
       - - - - -+- - - - +- - - - +- - - - +
              1 |       0 |       0 |      11 |    11
       - - - - -+- - - - +- - - - +- - - - +
       Total            4         19       11       34

                Table of smoke by smoke_ind
       Smoke      smoke_ind
       Frequency|       . |       0 |       1 | Total
       - - - - -+- - - - +- - - - +- - - - +
                |       1 |       0 |       0 |     1
       - - - - -+- - - - +- - - - +- - - - +
       current  |       0 |       0 |       8 |     8
       - - - - -+- - - - +- - - - +- - - - +
       never    |       0 |      18 |       0 |    18
       - - - - -+- - - - +- - - - +- - - - +
       past     |       0 |       7 |       0 |     7
       - - - - -+- - - - +- - - - +- - - - +
       Total            1        25         8       34
```


10.5 Using the OPTIONS Procedure to Modify SAS System Options

In Section 10.4.2, the FMTSEARCH= system option is used in the OPTIONS statement to search user-defined formats. In addition, SAS provides numerous different types of system options to control how the output are formatted, how the files are being handled, or how the data sets are being

processed, etc. To learn the current settings of a SAS system option, you can use the OPTIONS procedure, which has the following form:

```
PROC OPTIONS <LISTGROUPS><GROUP=><OPTION=>;
RUN;
```

Only a few options are listed in the syntax above. Other options can be found in SAS documentation. Without specifying any options, PROC OPTIONS will list all the system options in the SAS log. These system options can be categorized into different groups based on their functionality, and you can use LISTGROUPS to list the group names of all the system options. Once you know the name of each system option group, you can use the GROUP= option to display options belonging to one or more groups.

In Program 10.19, a few options are used in the OPTIONS statement to change the default system options. The NODATE option is used to suppress the date and time being printed in the listing output. NONUMBER is used to suppress the page number in the output. The LINESIZE= option sets the default line size to 65 characters per line in the SAS log and procedure output. The FORMDLIM= option specifies a blank line to delimit page breaks in the SAS output. All of these options are part of the LISTCONTROL system group and are listed by using the GROUP= option in the PROC OPTONS statement.

Program 10.19:

```
options nodate nonumber linesize = 65 formdlim = " ";
proc options group = listcontrol;
run;
```

Log from Program 10.19:

```
7    options nodate nonumber linesize = 65 formdlim = " ";
8    proc options group = listcontrol;
9    run;
   SAS (r) Proprietary Software Release 9.2 TS2M3

BYLINE             Print the byline at the beginning of each
                   by-group
CENTER             Center SAS procedure output
NODATE             Do not print date and time on top of each
                   page of SAS log and procedure output
NODETAILS          Do not display additional information in
                   directory lists
DMSOUTSIZE = 99999 Maximum number of rows in DMS output window
NODTRESET          Do not update date and time for log and print
                   output
FORMCHAR = ,ƒ"…†‡^‰Š‹Œ+ = |-/\<>*
                   Default output formatting characters for
                   print device
```

```
FORMDLIM =          Character to delimit page breaks in SAS
                    output
FORMS = DEFAULT     Default form used to customize appearance of
                    interactive windowing output
LABEL               Allow procedures to use labels with variables
LINESIZE = 65       Line size for SAS log and SAS procedure
                    output
MISSING =.          Character to represent missing numeric value
NONUMBER            Do not print page number on each page of SAS
                    output
NOPAGEBREAKINITIAL
                    Do not begin SAS log and listing files on a
                    new page.
PAGENO = 1          Beginning page number for the next page of
                    output produced by the SAS System
PAGESIZE = 56       Number of lines printed per page of output
NOPRINTINIT         Do not initialize SAS print file
SKIP = 0            Number of lines to skip before title
SYSPRINTFONT =      Set the default font for printing
HOSTPRINT           Print using the host Print Manager. To use
                    Universal Printing or FORMS, this option must
                    be turned off.
PRNGETLIST          Enable listing of the system printers. The
                    SAS system will discover printers installed
                    on the system.
SYSPRINT = ("Send To OneNote 2007")
                    Set the default printer and, optionally, an
                    alternate destination (file) for output
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Another useful option of the PROC OPTIONS statement is the OPTION= option, which is used to display a short description and the value of specified options. For example, Program 10.20 lists the LINESIZE option in the log.

Program 10.20:

```
proc options option = linesize;
run;
```

Log from Program 10.20:

```
231  proc options option = linesize;
232  run;
   SAS (r) Proprietary Software Release 9.2 TS2M0
LINESIZE = 65       Line size for SAS log and SAS procedure
                    output
```

```
NOTE: PROCEDURE OPTIONS used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Exercises

Exercise 10.1. Program 10.14 transposes the DAT4 data set to the DAT4_TRANSPOSE data set by using PROC TRANSPOSE twice. In this exercise, transpose the DAT4_TRANSPOSE data set back to the DAT4 data set.

Exercise 10.2. Based on the GRADE.SAS7BDAT data set, create three variables, MATH_POINT, ENGLISH_POINT, and PE_GRADE, using user-defined formats. The instructions for creating these three variables are described in *Exercise 2.1* in Chapter 2.