SAS TUTORIAL: TABLE LOOKUP TECHNIQUES

Don Henderson, ORI, Inc.

1. Problem Statement

This tutorial topic illustrates procedures for table lookup. Two major applications for table lookup are: 1) replacing a coded value with another value such as an alpha label; and 2) recoding, e.g., replacing some value or a range of values by a code, for example coding age in years into age groups (0-9, 10-19, etc.).

Various methods of performing table lookup in SAS are illustrated in the following sections. The first five examples operate on the data set used for the multiple output frequencies examples in the Transposing Data tutorial; for this topic, the variable REGION has been added to the data set. Methods for replacing REGION codes with names are presented. In the last section, several of the table lookup methods are combined to perform complex recoding.

2. Output Formats

In this example, the REGION code only needs to be replaced by its name on output (print). Therefore, it is not necessary to add a new variable to the data set. All that is required is the creation of a format library (lines 5-13, figure 1). A FORMAT statement (line 21, figure 1) is used to cause the REGION name to be printed instead of the REGION code itself.

3. IF Statements

IF statements can be used to add a new variable to the data set whose value is the REGION name. The IF statements used to create the variable REG_NAME (lines 4-13, figure 2) illustrate this technique.

There are two drawbacks associated with the use of IF statements for table lookup, especially for data sets with many distinct codes. First, the IF statements are tedious to code and tend to "clutter up" the program. Second, their use can be inefficient.

4. Merging Data with a Translation Table

The MERGE capability is frequently used for table lookup. It requires less coding than IF statements; however, it is also less efficient (in terms of machine time) and less flexible than most of the other methods presented here. Merging requires a separate sort and data step for each lookup, whereas the other methods allow for multiple table lookups in a single data step.

An example of table lookup using a MERGE is given in figures 3 and 4. A "translation" data set which contains one observation for each REGION code, with its name stored in the variable REG_NAME, is created (lines 6-9, figure 3). The data file is then sorted (lines 21-22, figure 3) and the data set is MERGEd with the translation data set (lines 23-24, figure 3) using REGION as a key. If the original sort of the data must be retained, another sort is required (lines 25-27, figure 3).

5. The PUT Function

The PUT function provides an efficient method of table lookup (figure 5). First, a format must be created (lines 5-13). In the subsequent data step (lines 15-20), the REGION name is added to the data set using the PUT function (line 19) which "writes" the formated value of the variable R into REGION. Note that REGION was renamed on the SET statement (line 17). This causes REGION to become a new variable in REDEFINE. It can then be defined as a character variable of length 12 (line 18) whose value is the region name obtained from the PUT function. Without the rename, it would not be possible to "change" REGION from a numeric variable to a character variable in a single data step.

If the translation table exists as a SAS data set rather than a format, the PUT function can still be used by first creating a format from that data set. An example is given in section 5 of the Selecting Subsets of Data tutorial.

6. Using ARRAY Structures

Another method of table lookup involves storing the labels/names in an array with the coded value used as the subscript into the array (figures 6 and 7). This method is particularly applicable if the codes are sequential numbers.

The translation data set is read in (lines 6-9, figure 6). In the next data step (lines 22-34, figure 6), the lookup is done. The ARRAY is defined (line 23) and the names are read into the ARRAY using a loop which is executed on the first observation (lines 25-31) with the values of the ARRAY elements retained (line 24) across all observations (note that the names could be read from an external file here; a prior data step to read them in is unnecessary). The observations are read from the data set and the lookup is done (lines 32-33). The value of REGION on each observation is used as the subscript into the ARRAY to get the corresponding value for REG_NAME.

The position of the code to read the names into the ARRAY is important. If this code had appeared after the main SET statement (line 32), the value of REGION on the first observation would be overwritten before the lookup. The lookup on the first observation would use the value of REGION from the last observation in TRNSLATE, resulting in an error.

7. Recoding Using the PUT Function and Arrays

This example (figures 8-10) illustrates a more complex table lookup application using ARRAYS and the PUT function. JOB_CODE and INDUS-

TRY must be recoded (lines 5-17, figure 8). This recoding can be done by storing the recoded values in a two-dimensional ARRAY with JOB_CODE and INDUSTRY used as subscripts into this ARRAY. This would require a large table, however. Instead, the job and industry codes can first be recoded, using the PUT function, to the row and column indices of the smaller 4x3 table. First, the formats to convert JOB_CODE and INDUSTRY to the row and column indices are created (lines 22-30, figure 8). The two-dimensional table lookup is done in one data step (lines 52-75, figure 9). The ARRAYs are defined (lines 54-58, figure 9) and the table values are read into the two-dimensional array once (lines 60-67, figure 9), retaining (line 59) the values of the ARRAY elements across all observations. The data to be recoded is read in (line 70); the lookups for the JOB_CODE and INDUSTRY indices are done (lines 72-73) using the PUT function; and finally, the two-dimensional recode/table lookup is done (line 74) using the ARRAYs.

```
1          *THIS EXAMPLE ILLUSTRATES THE USE OF FORMATS FOR TABLE
2           LOOKUP.  THE FORMATTED VALUE IS NOT ADDED TO THE DATA
3           SET, IT ONLY APPEARS AT PRINT TIME.;
4
5          PROC FORMAT;
6           *CREATE THE FORMATS;
7           VALUE REGFMT 1=NORTHEAST
8                        2='MID-ATLANTIC'
9                        3=SOUTH
10                       4=MIDWEST
11                       5=NORTHWEST
12                       6=WEST
13                       7=SOUTHWEST;

NOTE: THE PROCEDURE FORMAT USED 0.05 SECONDS AND 184K.

14         PROC FREQ DATA=SAVE.CNFRENCE;
15          TABLES REGION;
16          TITLE TABLE LOOKUP USING FORMATS;
17          TITLE2 FORMATTED VALUE ONLY APPEARS AT PRINT TIME;
18          TITLE3 WITHOUT THE FORMAT STATEMENT;

NOTE: THE PROCEDURE FREQ USED 0.17 SECONDS AND 172K
      AND PRINTED PAGE 1.

19         PROC FREQ DATA=SAVE.CNFRENCE;
20          TABLES REGION;
21          FORMAT REGION REGFMT.;
22          TITLE3 WITH THE FORMAT STATEMENT;

NOTE: THE PROCEDURE FREQ USED 0.18 SECONDS AND 172K
      AND PRINTED PAGE 2.
```


1

TABLE LOOKUP USING FORMATS
FORMATTED VALUE ONLY APPEARS AT PRINT TIME
WITHOUT THE FORMAT STATEMENT

REGION	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
1	157	157	15.907	15.907
2	131	288	13.273	29.179
3	141	429	14.286	43.465
4	145	574	14.691	58.156
5	133	707	13.475	71.631
6	142	849	14.387	86.018
7	138	987	13.982	100.000

TABLE LOOKUP USING FORMATS
FORMATTED VALUE ONLY APPEARS AT PRINT TIME
WITH THE FORMAT STATEMENT

REGION	FREQUENCY	CUM FREQ	PERCENT	CUM PERCENT
NORTHEAST	157	157	15.907	15.907
MID-ATLANTIC	131	288	13.273	29.179
SOUTH	141	429	14.286	43.465
MIDWEST	145	574	14.691	58.156
NORTHWEST	133	707	13.475	71.631
WEST	142	849	14.387	86.018
SOUTHWEST	138	987	13.982	100.000

FIGURE 1

```
1          *THIS EXAMPLE ILLUSTRATES THE USE OF IF STATEMNTS FOR TABLE
2           LOOKUP.  A NEW VARIABLE IS ACTUALLY ADDED TO DATA SET.;
3
4          DATA WITHNAME;
5           SET SAVE.CNFRENCE;
6           LENGTH REG_NAME $12;
7           IF REGION EQ 1 THEN REG_NAME = 'NORTHEAST';
8           ELSE IF REGION EQ 2 THEN REG_NAME = 'MID-ATLANTIC';
9           ELSE IF REGION EQ 3 THEN REG_NAME = 'SOUTH';
10          ELSE IF REGION EQ 4 THEN REG_NAME = 'MIDWEST';
11          ELSE IF REGION EQ 5 THEN REG_NAME = 'NORTHWEST';
12          ELSE IF REGION EQ 6 THEN REG_NAME = 'WEST';
13          ELSE IF REGION EQ 7 THEN REG_NAME = 'SOUTHWEST';
14

NOTE: DATA SET WORK.WITHNAME HAS 987 OBSERVATIONS AND 6 VARIABLES. 340 O
BS/TRK
NOTE: THE DATA STATEMENT USED 0.16 SECONDS AND 180K.

15         PROC PRINT DATA=WITHNAME(OBS=20);
16          TITLE TABLE LOOKUP USING IF STATEMENTS;
17          TITLE2 VARIABLE REG_NAME IS ADDED TO THE DATA SET;
18          TITLE3 PRINT OF 20 OBSERVATIONS FROM DATA SET WITHNAME;

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 172K
      AND PRINTED PAGE 1.

NOTE: SAS USED 180K MEMORY.
NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      BOX 8000
      CARY, N.O. 27511
```

1 TABLE LOOKUP USING IF STATEMENTS 1
VARIABLE REG_NAME IS ADDED TO THE DATA SET
PRINT OF 20 OBSERVATIONS FROM DATA SET WITHNAME

OBS	ID	PAPER	INVITED	ENJOY	REGION	REG_NAME
1	1	0	0	0	4	MIDWEST
2	2	0	0	0	3	SOUTH
3	3	1	0	1	4	MIDWEST
4	4	0	0	1	2	MID-ATLANTIC
5	5	0	0	1	6	WEST
6	6	1	0	0	1	NORTHEAST
7	7	0	0	1	1	NORTHEAST
8	8	1	0	0	3	SOUTH
2 9	9	0	1	1	5	NORTHWEST
10	10	0	0	1	2	MID-ATLANTIC
11	11	0	0	1	7	SOUTHWEST
12	12	0	0	1	2	MID-ATLANTIC
13	13	0	0	0	7	SOUTHWEST
14	14	0	0	0	2	MID-ATLANTIC
15	15	1	0	0	3	SOUTH
16	16	0	0	1	7	SOUTHWEST
17	17	1	0	1	4	MIDWEST
18	18	0	0	1	1	NORTHEAST
19	19	0	0	1	3	SOUTH
20	20	0	0	1	1	NORTHEAST

FIGURE 2

```
1          *THIS EXAMPLE USES THE MERGE CAPABILITY TO DO TABLE LOOKUP.
2           A "TRANSLATION DATASET IS READ IN AND MERGED WITH THE SORTED
3           MASTER FILE TO ADD THE VARIABLE REG_NAME.  UNLIKE THE OTHER
4           METHODS, ONLY ONE TABLE LOOKUP CAN BE DONE PER MERGE.;
5
6          DATA TRNSLATE;
7           LENGTH REG_NAME $12;
8           INPUT REGION REG_NAME $;
9           CARDS;

NOTE: DATA SET WORK.TRNSLATE HAS 7 OBSERVATIONS AND 2 VARIABLES. 794 OBS
/TRK
NOTE: THE DATA STATEMENT USED 0.04 SECONDS AND 172K.

17         PROC PRINT;
18          TITLE TABLE LOOKUP USING THE MERGE CAPABILITY TO ADD;
19          TITLE2 THE VARIABLE REG_NAME TO THE DATA SET.;
20          TITLE3 DATA SET TRNSLATE;

NOTE: THE PROCEDURE PRINT USED 0.08 SECONDS AND 172K
      AND PRINTED PAGE 1.

21         PROC SORT DATA=SAVE.CNFRENCE OUT=A;
22          BY REGION;

NOTE: 4 CYLINDERS DYNAMICALLY ALLOCATED PER SORT WORK DATA SET.
NOTE: DATA SET WORK.A HAS 987 OBSERVATIONS AND 5 VARIABLES. 433 OBS/TRK.
NOTE: THE PROCEDURE SORT USED 0.50 SECONDS AND 236K.

23         DATA AFTMERGE;
24          MERGE A TRNSLATE; BY REGION;

NOTE: DATA SET WORK.AFTMERGE HAS 987 OBSERVATIONS AND 6 VARIABLES. 340 O
BS/TRK
NOTE: THE DATA STATEMENT USED 0.20 SECONDS AND 180K.

25         PROC SORT;
26          *SORT BACK TO ORIGINAL ORDER;
27          BY ID;

NOTE: DATA SET WORK.AFTMERGE HAS 987 OBSERVATIONS AND 6 VARIABLES. 340 O
BS/TRK
NOTE: THE PROCEDURE SORT USED 0.29 SECONDS AND 236K.

28         PROC PRINT DATA=AFTMERGE(OBS=15);
29          TITLE3 FINAL DATASET - WITH REG_NAME ADDED.;

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 172K
      AND PRINTED PAGE 2.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      BOX 8000
      CARY, N.C. 27511
```

FIGURE 3

TABLE LOOKUP USING THE MERGE CAPABILITY TO ADD 1
THE VARIABLE REG_NAME TO THE DATA SET.
DATA SET TRNSLATE

OBS	REG_NAME	REGION
1	NORTHEAST	1
2	MID-ATLANTIO	2
3	SOUTH	3
4	MIDWEST	4
5	NORTHWEST	5
6	WEST	6
7	SOUTHWEST	7

TABLE LOOKUP USING THE MERGE CAPABILITY TO ADD 2
THE VARIABLE REG_NAME TO THE DATA SET.
FINAL DATASET - WITH REG_NAME ADDED.

OBS	ID	PAPER	INVITED	ENJOY	REGION	REG_NAME
1	1	0	0	0	4	MIDWEST
2	2	0	0	0	3	SOUTH
3	3	1	0	1	4	MIDWEST
4	4	0	0	1	2	MID-ATLANTIC
5	5	0	0	1	6	WEST
6	6	1	0	0	1	NORTHEAST
7	7	0	0	1	1	NORTHEAST
8	8	1	0	0	3	SOUTH
9	9	0	1	1	5	NORTHWEST
10	10	0	0	1	2	MID-ATLANTIC
11	11	0	0	1	7	SOUTHWEST
12	12	0	0	1	2	MID-ATLANTIC
13	13	0	0	0	7	SOUTHWEST
14	14	0	0	0	2	MID-ATLANTIC
15	15	1	0	0	3	SOUTH

FIGURE 4

```
1          *THIS EXAMPLE ILLUSTRATES THE USE OF FORMATS FOR TABLE
2           LOOKUP.  THE FORMATTED VALUE IS ADDED TO THE DATA SET AS THE
3           VALUE OF THE VARIABLE REGION WHICH IS REDEFINED AS CHARACTER.;
4
5          PROC FORMAT;
6           *CREATE THE FORMATS;
7           VALUE REGFMT 1=NORTHEAST
8                        2='MID-ATLANTIC'
9                        3=SOUTH
10                       4=MIDWEST
11                       5=NORTHWEST
12                       6=WEST
13                       7=SOUTHWEST;

NOTE: THE PROCEDURE FORMAT USED 0.05 SECONDS AND 184K.

15         DATA REDEFINE;
16          *RENAME USED SO "REGION" CAN BECOME A "NEW" VARIABLE.;
17          SET SAVE.CNFRENCE(RENAME=(REGION=R));
18          LENGTH REGION $ 12.;
19          REGION=PUT(R,REGFMT.);
20          DROP R;

NOTE: DATA SET WORK.REDEFINE HAS 987 OBSERVATIONS AND 5 VARIABLES. 397 O
BS/TRK
NOTE: THE DATA STATEMENT USED 0.16 SECONDS AND 180K.

22         PROC PRINT DATA=REDEFINE(OBS=15);
23          TITLE TABLE LOOKUP USING FORMATS AND THE PUT FUNOTION TO ADD;
24          TITLE2 THE FORMATTED VALUE OF REGION TO THE DATA SET.;
25          TITLE3 THE RENAME PARAMETER IS USED SO REGION DAN BE ;
26          TITLE4 DEFINED AS CHARACTER.;
27          TITLE5 PRINT OF FIRST 15 OBSERVATIONS IN DATA SET REDEFINE.;

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 172K
      AND PRINTED PAGE 1.
```

TABLE LOOKUP USING FORMATS TAND THE PUT FUNCTION TO ADD 1
THE FORMATTED VALUE OF REGION TO THE DATA SET.
THE RENAME PARAMETER IS USED SO REGION CAN BE
DEFINED AS CHARACTER.
PRINT OF FIRST 15 OBSERVATIONS IN DATA SET REDEFINE.

OBS	ID	PAPER	INVITED	ENJOY	REGION
1	1	0	0	0	MIDWEST
2	2	0	0	0	SOUTH
3	3	1	0	1	MIDWEST
4	4	0	0	1	MID-ATLANTIC
5	5	0	0	1	WEST
6	6	1	0	0	NORTHEAST
7	7	0	0	1	NORTHEAST
8	8	1	0	0	SOUTH
9	9	0	1	1	NORTHWEST
10	10	0	0	1	MID-ATLANTIC
11	11	0	0	1	SOUTHWEST
12	12	0	0	1	MID-ATLANTIC
13	13	0	0	0	SOUTHWEST
14	14	0	0	0	MID-ATLANTIC
15	15	1	0	0	SOUTH

FIGURE 5

```
1          *THIS EXAMPLE DOES TABLE LOOKUP BY READING IN A TRANSLATION
2           DATA SET AND THEN STORING THE INFORMATION IN AN ARRAY WHICH
3           IS RETAINED ACROSS ALL OBSERVATIONS IN THE DATA SET.  THE
4           VALUE OF REGION IS USED AS A SUBSCRIPT INTO THE ARRAY.;
5
6          DATA TRNSLATE;
7           LENGTH REG_NAME $12;
8           INPUT REGION REG_NAME $;
9           CARDS;

NOTE: DATA SET WORK.TRNSLATE HAS 7 OBSERVATIONS AND 2 VARIABLES. 794 OBS
/TRK
NOTE: THE DATA STATEMENT USED 0.04 SECONDS AND 172K.

17         PROC PRINT;
18          TITLE TABLE LOOKUP BY STORING THE VALUES IN AN ARRAY.;
19          TITLE2 REGION IS USED AS A SUBSCRIPT INTO THE ARRAY.;
20          TITLE3 PRINT THE THE TRANSLATION DATA SET WHICH WILL;
21          TITLE4 BE READ INTO THE ARRAY.;

NOTE: THE PROCEDURE PRINT USED 0.09 SEOONDS AND 172K
      AND PRINTED PAGE 1.

22         DATA USEARRAY;
23          ARRAY REGS (REGION) $ 12 R1-R7;
24          RETAIN R1-R7;
25          IF _N_ = 1 THEN
26          DO /*READ LABELS INTO THE ARRAY*/;
27            DO I=1 TO 7;
28               SET TRNSLATE POINT=I;
29               REGS=REG_NAME;
30            END;
31          END /*READ LABELS INTO THE ARRAY*/;
32          SET SAVE.CNFRENCE;
33          REG_NAME=REGS;
34          DROP R1-R7;

NOTE: DATA SET WORK.USEARRAY HAS 987 OBSERVATIONS AND 6 VARIABLES. 340 O
BS/TRK
NOTE: THE DATA STATEMENT USED 0.17 SEOONDS AND 180K.

35         PROC PRINT DATA=USEARRAY(OBS=15);
36          TITLE3 DATA SET USEARRAY WITH REG_NAME ADDED.;

NOTE: THE PROCEDURE PRINT USED 0.10 SECONDS AND 172K
      AND PRINTED PAGE 2.

NOTE: SAS USED 180K MEMORY.
NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      BOX 8000
      CARY, N.C. 27511
```

FIGURE 6

TABLE LOOKUP BY STORING THE VALUES IN AN ARRAY. 1
REGION IS USED AS A SUBSCRIPT INTO THE ARRAY.
PRINT THE THE TRANSLATION DATA SET WHICH WILL
BE READ INTO THE ARRAY.

OBS	REG_NAME	REGION
1	NORTHEAST	1
2	MID-ATLANTIC	2
3	SOUTH	3
4	MIDWEST	4
5	NORTHWEST	5
6	WEST	6
7	SOUTHWEST	7

TABLE LOOKUP BY STORING THE VALUES IN AN ARRAY. 2
REGION IS USED AS A SUBSCRIPT INTO THE ARRAY.
DATA SET USEARRAY WITH REG_NAME ADDED.

OBS	REGION	REG_NAME	ID	PAPER	INVITED	ENJOY
1	4	MIDWEST	1	0	0	0
2	3	SOUTH	2	0	0	0
3	4	MIDWEST	3	1	0	1
4	2	MID-ATLANTIC	4	0	0	1
5	6	WEST	5	0	0	1
6	1	NORTHEAST	6	1	0	0
7	1	NORTHEAST	7	0	0	1
8	3	SOUTH	8	1	0	0
9	5	NORTHWEST	9	0	1	1
10	2	MID-ATLANTIC	10	0	0	1
11	7	SOUTHWEST	11	0	0	1
12	2	MID-ATLANTIC	12	0	0	1
13	7	SOUTHWEST	13	0	0	0
14	2	MID-ATLANTIC	14	0	0	0
15	3	SOUTH	15	1	0	0

FIGURE 7

```
        *THIS EXAMPLE ILLUSTRATES THE USE OF FORMATS, THE PUT
         FUNCTION AND ARRAYS TO DO MORE COMPLEX TABLE LOOKUP. THE
         OBJECTIVE IS TO RECODE JOB CODE AND INDUSTRY AS FOLLOWS:

                             INDUSTRY
                      104-108,110   109    OTHER
                      --------------------------
                     |
               201-204,|
                 207   |    1         2       8
                       |
   JOB           205   |    3         4       8
   CODE                |
                 206   |    5         6       8
                       |
               OTHER   |    7         7       9
                       |

         THE PUT FUNCTION WITH FORMATS IS USED TO GET ROW AND COLUMN
         INDICES INTO AN ARRAY WHICH CONTAINS THE TABLE ENTRIES.;

        PROC FORMAT;
         *DEFINE THE ROW AND COLUMN INDIOES FORMATS;
         VALUE ROW 201-204,207=1
                          205=2
                          206=3
                        OTHER=4;
         VALUE COL 104-108,110=1
                          109=2
                        OTHER=3;

NOTE: THE PROCEDURE FORMAT USED 0.08 SECONDS AND 184K.

        DATA JOBS;
         *READ IN THE DATA TO BE RECODED;
         INPUT JOB_CODE INDUSTRY;
         CARDS;

NOTE: DATA SET WORK.JOBS HAS 10 OBSERVATIONS AND 2 VARIABLES. 953 OBS/TR
K
NOTE: THE DATA STATEMENT USED 0.04 SECONDS AND 172K.

        PROC PRINT;
         TITLE THE USE OF FORMATS, THE PUT FUNCTION AND ARRAYS TO;
         TITLE2 DO MORE COMPLICATED TABLE LOOKUP.  THE TABLE IS;
         TITLE3 READ INTO A TWO DIMENSIONAL ARRAY.;
         TITLE4 LISTING OF THE INPUT DATA.;

NOTE: THE PROCEDURE PRINT USED 0.08 SECONDS AND 172K
      AND PRINTED PAGE 1.
```

FIGURE 8

```
52          DATA RECODE;
53           *SET UP THE ARRAYS FOR THE TABLE;
54           ARRAY ROW1 (C) X11-X13;
55           ARRAY ROW2 (C) X21-X23;
56           ARRAY ROW3 (C) X31-X33;
57           ARRAY ROW4 (C) X41-X43;
58           ARRAY TABLE (R) ROW1-ROW4;
59           RETAIN X11--X43;
60           IF _N_=1 THEN
61           DO /* READ IN THE TABLE */;
62            DO R = 1 TO 4;*READ EACH ROW;
63             DO C=1 TO 3;*READ EACH COLUMN;
64               INPUT TABLE @;
65             END;
66            END;
67           END /* READ IN THE TABLE */;
68
69
70           SET JOBS;
71           *READ IN THE DATA TO BE RECODED;
72           R=PUT(JOB_CODE,ROW.);
73           C=PUT(INDUSTRY,COL.);
74           NEW_CODE=TABLE;
75          CARDS;

NOTE: CHARACTER VALUES HAVE BEEN CONVERTED TO NUMERIC
      VALUES AT THE PLACES GIVEN BY: (LINE):(COLUMN).
      72:4  73:4
NOTE: SAS WENT TO A NEW LINE WHEN INPUT STATEMENT
      REACHED PAST THE END OF A LINE.
NOTE: DATA SET WORK.RECODE HAS 10 OBSERVATIONS AND 17 VARIABLES. 136 OBS
/TRK
NOTE: THE DATA STATEMENT USED 0.10 SECONDS AND 180K.

80          PROC PRINT DATA=RECODE(OBS=1);
81           VAR X11--X43;
82           TITLE4 THE TABLE;

NOTE: THE PROCEDURE PRINT USED 0.09 SECONDS AND 172K
      AND PRINTED PAGE 2.

83          PROC PRINT;
84           DROP X11--X43;
85           TITLE4 DATA RECODE - WITH THE VARIABLE NEWCODE;

NOTE: THE PROCEDURE PRINT USED 0.09 SECONDS AND 172K
      AND PRINTED PAGE 3.

NOTE: SAS INSTITUTE INC.
      SAS CIRCLE
      BOX 8000
      CARY, N.C. 27511
```

FIGURE 9

THE USE OF FORMATS, THE PUT FUNCTION AND ARRAYS TO DO MORE COMPLICATED TABLE LOOKUP. THE TABLE IS READ INTO A TWO DIMENSIONAL ARRAY. 1

LISTING OF THE INPUT DATA.

OBS	JOB_CODE	INDUSTRY
1	201	104
2	205	108
3	206	105
4	210	109
5	207	109
6	300	99
7	205	109
8	206	200
9	100	109
10	206	109

THE USE OF FORMATS, THE PUT FUNCTION AND ARRAYS TO DO MORE COMPLICATED TABLE LOOKUP. THE TABLE IS READ INTO A TWO DIMENSIONAL ARRAY. 2

THE TABLE

OBS	X11	X12	X13	X21	X22	X23	X31	X32	X33	X41	X42	X43
1	1	2	8	3	4	8	5	6	8	7	7	9

THE USE OF FORMATS, THE PUT FUNCTION AND ARRAYS TO DO MORE COMPLICATED TABLE LOOKUP. THE TABLE IS READ INTO A TWO DIMENSIONAL ARRAY. 3

DATA RECODE - WITH THE VARIABLE NEWCODE

OBS	C	R	JOB_CODE	INDUSTRY	NEW_CODE
1	1	1	201	104	1
2	1	2	205	108	3
3	1	3	206	105	5
4	2	4	210	109	7
5	2	1	207	109	2
6	3	4	300	99	9
7	2	2	205	109	4
8	3	3	206	200	8
9	2	4	100	109	7
10	2	3	206	109	6

FIGURE 10