Getting the Most from PROC TABULATE
Andrew A. Norton, ORI, Inc.

I. Why use PROC TABULATE?

PROC TABULATE is more difficult to master than the typical SAS procedure. Many people have learned alternative methods of producing complex reports, such as PROC PRINT or DATA steps. It is reasonable to ask what advantages of the procedure makes it worth learning.

The most distinctive advantage of PROC TABULATE is that it dynamically adjusts to the data. If a new value of a classification variable is added to a report PROC TABULATE adjusts automatically. The new values are added as new rows and columns, in their correct positions. If the size of the table expands sufficiently, PROC TABULATE will automatically break it into sections so that each section will fit on a page.

This dynamic expansion capability allows the table to be specified in abstract terms, so that the computer code remains applicable even if the categories change. The exact shape and dimensions of the table are determined using the input data.

This separation of the specification of the logical structure of a table from the details of a particular dataset is paralleled by the separation of the specification of the logical structure of the table from the labeling and presentation of the table. User-defined and system-supplied formats may be used for cell values and classification values, and labels may be supplied for any kind of header. These cosmetic refinements of the table are very flexible and easy to modify because they are independent of the fundamental table design.

It is easy to make sweeping changes in the appearance and structure of a PROC TABULATE table by only slightly altering the code. The syntax is very concise, and many meaningful and useful variations of the same basic table may be produced. There is no need to change many different parts of the code simultaneously. Contrast this with a table-generating program using PROC SUMMARY in combination with PROC PRINT. Both procedures would typically have to be changed to produce a meaningful end result. This is because PROC SUMMARY computes the numbers, then PROC PRINT produces the report. In PROC TABULATE, by contract the computation and reporting phases are packaged together. A single specification of the table controls both phases. Changing the PROC TABULATE code changes both the computations and the design of the corresponding report. This feature is a key reason why PROC TABULATE code is easy to modify.

II. The Table-Development Process

While PROC TABULATE tables are easy to modify, they are tricky to set up from scratch. Many people search the extensive documentation in vain, trying to figure out how to use the complex syntax to produce a particular table, perhaps trying to copy an existing table or to follow a detailed design specification. There are two problems with this approach. The first is that PROC TABULATE is a package (albeit a powerful one) rather than a full-fledged programming language. There are many table designs that PROC TABULATE simply cannot accommodate. The procedure can probably produce a table that effectively presents the information, but if a particular table design is required, it will probably have to be custom-programmed using PUT statements in a DATA step.

The second problem encountered is that the solution to producing a table is often found not in the syntax of PROC TABULATE, but in the structure of the dataset input to the procedure.

As mentioned above, PROC TABULATE combines two phases: computation and reporting. As with PROC SUMMARY, a particular data set structure is required for the procedure to work. The computation capabilities are similar to those of PROC SUMMARY. Each observation of the input dataset is classified into a cell defined by classification variables; then a statistic is computed on an analysis variable across all observations in each cell.

The PROC TABULATE syntax is the wrong place to search for answers to problems in the computation phase, because the syntax deals primarily with problems in the reporting phase. Obviously, it is not possible to report numbers that have not been computed, so the first step in the development of any table is to get PROC TABULATE to compute the numbers you want in your table. The solution to computation phase problems usually involves modifications of the input dataset.

III. The development cycle of a PROC TABULATE table

The author’s experience has been that when development of a PROC TABULATE table has gone well, it has generally followed a logical sequence which dealt with the fundamental problem of computing the numbers first, then the layout of the numbers in the table, and finally the labels, formats, and other cosmetic details. Each step is easier to handle if the

---

*SAS is a registered trademark of SAS Institute Inc., Cary, N.C., USA.
The development of PROC TABULATE tables tends to be an iterative process based on trial and error. Except in trivial cases, the code will actually have to be run to evaluate the appearance of the table, and several stages of refinement will probably be necessary. Interactive SAS is very useful at this point, using a small test dataset (limiting the range of values of the classification variables to keep the table compact).

The remainder of this paper focuses on the development of the computation phase of PROC TABULATE tables. After some fundamental concepts are introduced, the computation processing of PROC TABULATE will be described. A repertoire of methods for handling problems involving the computation phase will be reviewed. The resolution of these problems usually requires transformation of the input data before PROC TABULATE even begins execution.

This approach to the development of PROC TABULATE tables has been infrequently described. While PROC TABULATE is the final stage, the groundwork is laid by DATA steps and procedures such as PROC SUMMARY, which prepare the input data.

IV. Conceptual Framework

Three concepts are essential to the remainder of this paper: "Analysis variables", "Class variables", and "Subtables".

Analysis Variables

The variables from the input dataset used by PROC TABULATE are declared as either "analysis" or "class" variables. The statistics are calculated on analysis variables, which are numeric variables declared in the VAR statement. There may be several analysis variables within the table, but in any one cell the procedure calculates one statistic on one analysis variable.

The statistics available are the same as in PROC MEANS, plus percentages. There are no distributional statistics such as means, medians, or percentiles.

Missing values of analysis variables are no problem. The missing value does not affect any statistic except NMISS, which is provided to count missing values.

Formats may be specified for analysis variables. They take effect after the computation is completed. These formats may include collapsing or individual value labels as well as continuous numeric formats.

A variable cannot be defined or used as both analysis and class in the same execution of PROC TABULATE -- SAS would get confused.

Class Variables

Class (or classification) variables are declared in the CLASS statement. These variables classify each observation according to the values of the class variables and place it in a group with any other input observations with these values. Only the first 16 characters are used. If a class variable values differ past the 16th character, they will be grouped together. Numeric class variables are treated as character.

If a format is specified, classification is based on the first 16 characters of the formatted value, rather than the original value.

If the MISSING option is not specified (this is the default), all observations with any class variable missing are completely removed from the tabulation. This occurs even if the variable does not apply to this particular table or computation in the table.

There is no way to keep some missing values but not others or to use observations in only sections of the table where the missing values don’t matter. For example, if a table has a grand total, and detail totals for each value of a class variable, the grand total will not include observations with missing values for the class variable.

The advantage of this missing value screen is consistency between different parts of the table or several tables specified in the same execution of the procedure. (This is the best reason to have more than one TABLE statement in the same execution of PROC TABULATE -- the tables will be consistent with each other).

The disadvantage is that sections of the table will not be the same as if they had been tabulated separately. If this is unacceptable, the missing option can be specified to treat missing values like any other values (special missing values remain distinct). Each section of the table will then be unaffected by the contents of other sections of the table.

Subtables

PROC TABULATE tables may have very complex structures, but they are always made up of one or more "subtables", the basic component of a table. This term is not used in the SAS manuals, but the concept has proven to be very useful.
A subtable consists of a set of cells which share the same set of class variables, the same statistic, and the same analysis variable. Each subtable uses each and every observation in the input database exactly once (except those removed by the missing value screen). Each observation is classified by class variable values into exactly one cell in the subtable. The statistic is calculated on the analysis variable across the observations in each cell. Every cell that contains data must be printed in the table. Subtable cells without data may be printed, depending on the format of the table.

Not only must every observation in the database be classified into exactly one cell of each subtable, but every observation must be reported with the same level of detail. It is not possible, for example, for PROC TABULATE to break down information on 4-year colleges in more detail than 2-year colleges.

Tables composed of multiple subtables

Tables may be composed of several concatenated subtables. These concatenations are identified by a concatenation operator in TABULATE syntax (2 or more analysis variables, class variables, or statistics separated by a blank rather than an asterisk, parenthesis, or comma). Concatenations may be difficult to detect by inspecting the table produced by PROC TABULATE, because the cells of a subtable are not necessarily all adjacent in the printed table. For example, the concatenation may be inside a nesting, so the rows or columns of the subtable are spread throughout the table. In every case, the underlying subtable structure still remains.

Note that there is no claim that the subtable structure has any effect on computational efficiency. This issue has not been investigated. The concept of subtables, for the purpose of this paper, is an analytical tool for developing PROC TABULATE tables.

"summary" subtables

One of the most useful concatenations involves one subtable which uses the same definition as another except for the omission of one or more class variables. The "ALL" operator may be used in their place, acting as a syntactical placekeeper to control the structure of the table. This concatenation gives the effect of a summarization of a detailed subtable. The "summary" subtable is, in fact, a separate subtable which could be tabulated by itself (although the results may differ because of missing class variable values as discussed above). This subtable may be printed interleaved with the detail subtable, to give the appearance of "subtotals," or all the cells may be grouped together.

This "summary" effect can only be achieved by PROC TABULATE when summarizing across the values of a class variable. It is not a direct summarization of cells in the table. It merely gives that effect by matching one subtable to another which is identical except for omitted class variables. Since subtables are made up of cells differentiated only by the values of class variables, this gives the effect of collapsing a set of cells from the detailed subtable.

It is not possible for PROC TABULATE to summarize across other types of cells. That is, the procedure cannot summarize across cells from more than one subtable. In particular, PROC TABULATE cannot summarize across different analysis variables. For example, you cannot have totals for different days (each an analysis variable) and then a total for the week.

V. The Computation Processing of PROC TABULATE

This section is intended to describe the operation of PROC TABULATE during the computation phase. It does not necessarily describe the way in which this functioning is actually implemented in SAS code.

1) If the MISSING option is not specified (the default condition), then remove every observation which has a missing value for any variable declared in the CLASS statement.

2) Parse the TABLE statement to determine the subtables. Each concatenation operator defines additional subtables. Commas (used to indicate dimensions) are considered for this purpose to be nestings. For this reason, the definition of any dimension remains fixed throughout the expansion of other dimensions (for example, the column heading cannot be changed halfway through the pages).

For each subtable, determine the class variables, statistic, and analysis variable. The "ALL" operator is not a class variable; it is used for parsing only.

3) For each class variable format each value and consider only the first 16 characters to find all values which occur in the dataset.

4) Determine the cells of each subtable by taking all combinations of the values of the class variables for that subtable. The limitations of PROC TABULATE are not clearly determined. There appears to be a limit of 32,767 rows (including rows with no observations). It is very easy to exceed this limit when several many-valued class variables are nested, as combinations which do not occur in the data count against the limit. Caution should be used with tables exceeding 32,767 cells (all combinations of all class variables), because incorrect results may be printed with no warning (as of SAS OS/MVS version 5.03).

5) Assign each observation to one cell of each subtable according to the values of the class variables. In each subtable, each observation must be classified into one and only one cell.
6) For each cell of each subtable, compute the statistic on the analysis variable over all observation in that cell. Set a flag to distinguish between missing values resulting from all observations in the cell having missing values on the analysis variable, and missing values resulting from an empty cell with no observations. This flag will be used in the reporting phase.

7) Continue to the reporting phase. Print the cells using the table structure defined by the TABLE statement.

VI. Input Data Structure for PROC TABULATE

From the point of view of PROC TABULATE, each observation consists only of class variables, used for classification into cells, and analysis variables, used for statistical computation within those cells. The art of developing PROC TABULATE tables involves extending the capabilities of the procedure by preparing the input dataset to achieve the desired numbers, then using the labeling capabilities of PROC TABULATE to make it appear as if some other computation had actually taken place. For example, the author's 1985 SUGI paper "PROC TABULATE Applications" discusses how to use the procedure to present statistics such as medians which are not supported by the procedure. Computations are completed prior to PROC TABULATE. A dataset is then constructed with one observation per cell and the medians as values of an analysis variable. PROC TABULATE actually computes the mean of the analysis variable, but the table is relabeled to make it appear as if a median had been computed.

VII. Techniques

The techniques described below fall in three categories: adding constructed variables, adding constructed observations, and reshaping the input dataset. They are a starting point for the development of a wide variety of PROC TABULATE tables. In many cases, the techniques should be used in combination to achieve the desired results.

Constructed Variables

Many times, all that is needed is the construction of an additional variable. A simple application uses the same variable as both an analysis and class variable. This would confuse the PROC TABULATE parser, but it is easy to create a copy of the variable with a different name, and use one name for class variable references and the other for analysis variable references. Remember that adding a class variable copy of an analysis variable makes the observations subject to the missing value screen on that variable. This method is useful for ordinal-level data, such as questionnaire items of the form "rate this on a scale of 1 to 5." Example 1 presents such an application.

Limitations on the number of cells can frequently be circumvented by concatenating two or more classification variables. This limitation is often encountered when cross-tabulating transitions from one state to another or transactions between categories. For example, students may have a choice of 200 fields of study, and may switch from one to another. PROC TABULATE would consider a table of changes this to have 200 x 200 = 40,000 cells, even though most of them are empty and not printed. This report can be produced by concatenating the two "field of study" variables together, so that only combinations actually occurring in the dataset are considered by PROC TABULATE as cells.

Because PROC TABULATE only distinguishes class variables by the first 16 characters, values such as "WASHINGTON STATE UNIVERSITY" and "WASHINGTON STATE COMMUNITY COLLEGE SYSTEM" are considered to be identical. Values longer than 16 characters should be coded using a unique abbreviation, and a format should be used to display the full name.

The order in which the class variable values are printed can be controlled by using the ORDER=INTERNAL option and user-defined formats. The variable should be recoded so that the internal values of the data are in the correct sort order. The format should be set up to print the desired value in the table. For example, days of the week could be coded as '1' to '7' internally, and formatted to 'SUNDAY' through 'SATURDAY'.

As described above, any value can be inserted into a cell by placing it in an analysis variable, then computing a MEAN on the analysis variable of that observation.

One of the most useful constructed variables is an analysis variable which is '1' if a condition is true, '0' if it is false. This can be used in a sum to obtain frequency counts which might otherwise be difficult or impossible to obtain. Computing a mean gives proportions. In either case, the information is presented in one cell rather than the two that would be required if this variable was used as a classification variable. This technique is fundamental to the tabulation of questions that allow multiple responses. A variable is constructed for each possible response, indicating its presence or absence. Example 3 in the SAS version 6 manual demonstrates this technique.

Constructed variables can be used to bypass the missing value screen. For example, a grand total or average may include observations which cannot be tabulated in more detailed subtables because of missing class variables. Construct a variable which is missing for observations with missing class variable values, and otherwise has the original analysis variable value. Replace the missing variable values with legitimate
values found in the input data so that no new class variable values are introduced. Use the constructed analysis variable for the detailed subtable and the original variable for the grand mean. Example 2 illustrates this technique.

Adding constructed observations

A great deal of control over tables can be achieved by adding observations to the dataset. In effect, two or more different datasets can be used for different parts of the same table. The key is to use analysis variables which are set to missing for all observations which are not involved in that section of the table. The technique is not as easy to use as it might appear, because every class variable on each observation must be set to a value which does not disturb the structure of the table. Class variables on each observation must be nonmissing so that the observation is not excluded by the missing value screen. Furthermore, they must be set to values that legitimately occur in the data, otherwise the observation will introduce new classification variable values into the table. This problem occurs even if the observation has missing values on all analysis variables used in cells defined by that classification variable. The values used to fill these classification variables may be values known to always occur in the input data, or the values may be obtained by using a DATA step or PROC FREQ to scan the original input data and obtain a value.

One use of constructed observations is to ensure that a fixed set of classification variable values appears in the table. Normally, the data-driven tables produced automatically by the procedure are advantageous, but there are times when class variable values are desired in the report even though they do not appear in a particular input dataset. For example, there may be a requirement that a report always reference each day of the week or each employee on a project (even if the employee was on vacation that week). If a observation for each desired value of the class variable is added to the dataset, these values will always appear in the data and be presented in the table. Appropriate values must be used to fill the other classification variables. Depending on the structure of the table and the options selected, it may be necessary to add observation for combinations of class values to be sure those cells are printed.

By default, class values are presented in alphabetical order. In the case of days of the week, this would have "FRIDAY" followed by "MONDAY" then "SATURDAY". One alternative is to recode the variable and use the ORDER=INTERNAL option discussed above. Another alternative is to use the ORDER=DATA option, which specifies that PROC TABULATE should use the order in which the values are found in the data. The value observations described above can be included at the beginning of the dataset to set the order. This alternative does not require a format and can be used to set the order of thousands of class variable values. Unfortunately, the ORDER option applies to every class variable in the table. This gets messy because sort order observations usually have to be constructed for every class variable, and the fill values for each class variable must be the first value (in sort order) rather than just any valid value. Example 3 shows a staff report program using such techniques.

When using the ORDER=DATA option, beware of simply sorting the original input dataset by the class variables unless it is certain that every combination of class variable values is always present in the input data. If this is not true, the values will probably not be encountered in the data in correct sort order, because of the missing cells.

The N statistic (without an analysis variable) counts all observations, including sort order observations. To get correct frequency counts, construct an analysis variable equal to 1 for the data observations and 0 for the added observations, then do a SUM.

Reshaping the dataset

Reshaping techniques change the organization of the original input data rather than simply adding observations or variables. These powerful techniques include transposition, assembling several observations into a single observation, splitting observations into several observations, sorting, and deleting observations.

PROC TABULATE is designed to compute statistics across a group of observations. It cannot compute statistics across a group of variables. For example, consider a dataset with one observation per state, and one variable per year. It is easy to find the mean value per year across the states, but PROC TABULATE cannot find the mean value per state across the years. PROC TRANSPOSE could be used to produce a dataset (with one observation per year, and one variable per state) which could be used to produce such a table.

A better solution would be to use PROC TRANSPOSE with a BY statement or a DATA step with arrays and OUTPUT statements to produce a dataset with one observation per state per year. Such a dataset could be used to classify observations by state, by year, or state and year, thereby producing a single table with state averages and year averages.

In general, it is a good idea to use observations identified with a combination of class variables at a very detailed level, rather than using analysis variables to separate related information. Each level of analysis is thus stored in its own dataset. With such data organization, PROC TABULATE can dynamically adapt to the class variable values contained in the input data without need to change the code. In the example above, if YEAR was a class variable, the program would not need to be modified next year when another year of data becomes available.
There are times, however, when it is desirable to transpose a set of observations (for different values of a class variable) into a single observation with several corresponding analysis variables. This makes it possible to display statistics for some values but not others, to easily control value order, and to keep a fixed set of cells present. This reshaping can be accomplished using arrays, and a SET statement controlled with a BY statement. These variables can be declared as classification variables to produce cross-tabulations of the frequency of various combinations. For example, observations for each state with variables for each year could be used to produce tables showing changes from one year to the next. This kind of table is only possible if the values for each year are stored on the same observation.

Another way of reshaping the input dataset is to delete cases, explicitly performing the missing value screen. Use the MISSING Option to turn off the standard screen.

Yet another way of manipulating the input data is to sort the input dataset and use a BY statement, which in effect runs PROC TABULATE individually within each by-group. The BY statement provides a clean way of controlling the expansion of class variables into cells. Only those class variable values appearing on observations in that by-group are used. If the by-variable defines a fundamental division between observations, use of the BY statement can produce much more clear and concise tables.

VIII. Conclusion

This potentially bewildering array of techniques encompasses many examples of the same basic strategy. In developing PROC TABULATE tables, distinguish between computing and reporting problems. Work with the input dataset to solve the former, and PROC TABULATE syntax to resolve the latter. To enable the PROC TABULATE syntax to do its job, add variables, add observations, and reshape the input dataset. Finally, relabel the resulting table to effectively and attractively present the statistical results. This effort will be repaid by tables which will automatically adjust to future conditions without code modifications.

The author can be contacted at:
GR1, Inc.
601 Indiana Avenue, N.W.
Suite 1000
Washington, D.C. 20004
(202) 737-2666

***************
/* EXAMPLE 1. */
/* ORDINAL DATA */
***************
/* RATING ON A SCALE OF 1 TO 5 */

DATA TABINPUT;
/* DATASET CONSTRUCTED FOR INPUT TO PROC TABULATE */
SET RATINGS;
/* USE RATING AS ANALYSIS VAR */
/* CREATE COPY FOR CLASS VAR */
CLASSVAR = RATING;
RUN;

PROC TABULATE DATA=TABINPUT
FORMCHAR=' :----1+1---';
TITLE 'RATINGS OF THE ITEMS';
VAR RATING;
CLASS CLASSVAR ITEM;
TABLE ITEM,
THE BLANK LABELS MUST BE IN THE COLUMN DIMENSION TO WORK PROPERLY */
CLASSVAR='RATING ' N= 1 ' F=5.0
RATING='MEAN ' MEAN=' N= 1 ' F=7.2
RATING='N ' N= 1 ' F=4.0
RTS=14 MISSTEXT='0';

RATINGS OF THE ITEMS

<table>
<thead>
<tr>
<th>ITEM</th>
<th>RATING</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>MEAN</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.33</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3.33</td>
<td>3</td>
</tr>
</tbody>
</table>
/* EXAMPLE 2 */
/* SELECTIVE EXCLUSION OF OBSERVATIONS */
/* WITH MISSING CLASS VARIABLES */

/* THIS EXAMPLE DEMONSTRATES HOW TO PRODUCE A TABLE WHICH INCLUDE ALL OBSERVATIONS IN THE GRAND MEAN, BUT ONLY VARIABLES WITH NONMISSING CLASS VARIABLE VALUES IN THE DETAILED CELLS USING THOSE CLASS VARIABLES */

INPUT DATASET COLLEGES
OBS GPA CONTROL TYPE
1 2.8 PUBLIC 2-YEAR
2 2.6 PRIVATE 4-YEAR
3 2.5 PUBLIC
4 2.2 4-YEAR

DATA TABINPUT (KEEP = TYPE CONTROL GPA TYPEGPA CNTRLGPA);
LENGTH FILLTYPE $7 FILLCNTL $8;
RETAIN FILLTYPE FILLCNTL;
IF _N_ EQ 1 THEN DO; /* GET FILL VALUE FOR TYPE */
   DO UNTIL (FILLTYPE NE ' ' AND FILLCNTL NE ' ');
      SET COLLEGES (KEEP=TYPE CONTROL);
   IF TYPE NE ' ' THEN FILLTYPE = TYPE;
   IF CONTROL NE ' ' THEN FILLCNTL = CONTROL;
   END;
END;
SET COLLEGES;
IF TYPE EQ ' ' THEN /* TYPEGPA WILL BE MISSING */
   TYPE = FILLTYPE;
ELSE
   TYPEGPA = GPA;
   IF CONTROL EQ ' ' THEN /* CNTRLGPA WILL BE MISSING */
      CONTROL = FILLCNTL;
   ELSE
      CNTRLGPA = GPA;
   /* NOW NO COLLEGE HAS A MISSING CLASS VARIABLE, SO NONE WILL BE SCREENED OUT */
RUN;

OBS GPA CONTROL TYPE TYPEGPA CNTRLGPA
1 2.8 PUBLIC 2-YEAR 2.8 2.8
2 2.6 PRIVATE 4-YEAR 2.6 2.6
3 2.5 PUBLIC 2-YEAR 2.5
4 2.2 PUBLIC 4-YEAR 2.2

(continued on next page)
Example 2 continued

PROC TABULATE DATA=TABINPUT FORMCHAR='|---|+|---';
TITLE 'GPA BY TYPE AND CONTROL OF COLLEGE';
CLASS TYPE CONTROL;
VAR GPA TYPEGPA CNTRLGPA;
TABLE TYPE * TYPEGPA = 'GPA'
          CONTROL * CNTRLGPA = 'GPA'
          ALL * GPA,
          MEAN*F=5.1 N*F=5.0;
RUN;

GPA BY TYPE AND CONTROL OF COLLEGE
-----------------------------------
<table>
<thead>
<tr>
<th></th>
<th>MEAN</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>---</td>
</tr>
<tr>
<td>2-YEAR</td>
<td>GPA</td>
<td>2.8</td>
</tr>
<tr>
<td>14-YEAR</td>
<td>GPA</td>
<td>2.4</td>
</tr>
<tr>
<td>CONTROL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIVATE</td>
<td>GPA</td>
<td>2.6</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>GPA</td>
<td>2.6</td>
</tr>
<tr>
<td>ALL</td>
<td>GPA</td>
<td>2.5</td>
</tr>
</tbody>
</table>
/* Example 3 */
/* Staff Report */

DATA HOURS;
  INPUT EMPLOYEE $8. HRS 2. DAY $3.;
CARDS;
ABBOTT 4TUE
COSTELLO 3WED
COSTELLO 2WED
ABBOTT 5THU
ABBOTT 3THU
COSTELLO 2THU
RUN;

PROC PRINT DATA=HOURS;
  TITLE 'INPUT DATASET HOURS';
RUN;

PROC FREQ DATA=HOURS;
  TABLES EMPLOYEE,OUT=EMPLOYEE (KEEP=EMPLOYEE) NOPRINT;
RUN;

PROC FREQ DATA=HOURS;
  TABLES EMPLOYEE/OUT=EMPLOYEE (KEEP=EMPLOYEE) NOPRINT;
/* NOTE THE OUTPUT FROM PROC FREQ IS ALREADY SORTED */
RUN;

DATA DAY;
  INPUT @1 DAY $3.;
CARDS;
  SUN
  MON
  TUE
  WED
  THU
  FRI
  SAT
RUN;

(continued on next page)
Example 3 continued

/* CONSTRUCT DATASET FOR INPUT TO PROC TABULATE */
DATA TABINPUT (KEEP= EMPLOYEE DAY HRS);
IF _N_ EQ 1 THEN
  DO;
    SET EMPLOYEE (KEEP=EMPLOYEE RENAME=(EMPLOYEE= EMPFILL));
    SET DAY (KEEP=DAY RENAME=(DAY = DAYFILL));
  END;
/* IT IS NOT NECESSARY TO EXPLICITLY RETAIN THE FILL VARIABLES
BECAUSE THE SET STATEMENTS ARE ONLY EXECUTED ONCE */
SET EMPLOYEE (IN=INE)
  DAY (IN=IND)
* FILL CLASS VARIABLES IN SORT ORDER RECORDS;*
  IF INE THEN EMPLOYEE = EMPFILL;
  IF IND THEN DAY = DAYFILL;
RUN;

/* PRINT TABINPUT DATASET FOR REVIEW */
PROC PRINT DATA=TABINPUT;
  TITLE 'CONSTRUCTED DATASET TABINPUT';
RUN;

CONSTRUCTED DATASET TABINPUT

<table>
<thead>
<tr>
<th>OBS</th>
<th>EMPLOYEE</th>
<th>DAY</th>
<th>HRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABBOTT</td>
<td>SUN</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>ABBOTT</td>
<td>SUN</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>ABBOTT</td>
<td>SUN</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>ABBOTT</td>
<td>MON</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ABBOTT</td>
<td>TUE</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>ABBOTT</td>
<td>WED</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>ABBOTT</td>
<td>THU</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>ABBOTT</td>
<td>FRI</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>ABBOTT</td>
<td>SAT</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>ABBOTT</td>
<td>TUE</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>COSTELLO</td>
<td>WED</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>COSTELLO</td>
<td>WED</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>ABBOTT</td>
<td>THU</td>
<td>12</td>
</tr>
<tr>
<td>14</td>
<td>ABBOTT</td>
<td>THU</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>COSTELLO</td>
<td>THU</td>
<td>14</td>
</tr>
</tbody>
</table>

PROC TABULATE DATA=TABINPUT ORDER=DATA FORMCHAR=' 1----1+1---';
  TITLE 'STAFF REPORT';
  CLASS EMPLOYEE DAY;
  VAR HRS;
  TABLE EMPLOYEE ALL, (DAY ALL) * SUM*HRS*F=5.0
    /RTS=12 MISSTEXT='0';
RUN;

<table>
<thead>
<tr>
<th>STAFF REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY</td>
</tr>
<tr>
<td>SUN</td>
</tr>
<tr>
<td>SUM</td>
</tr>
<tr>
<td>HRS</td>
</tr>
<tr>
<td>EMPLOYEE</td>
</tr>
<tr>
<td>ABBOTT</td>
</tr>
<tr>
<td>COSTELLO</td>
</tr>
<tr>
<td>ALL</td>
</tr>
</tbody>
</table>

127