Software engineering using the SAS System is much more than just writing code that is free of syntax errors. Today's software applications are very diverse, requiring knowledge of a broad range of sophisticated processing techniques. The vast capabilities of SAS software can simplify the implementation of many complicated data processing tasks, as long as system designers and programmers are able to identify useful and efficient techniques. This tutorial offers a variety of algorithms and routines — some fairly complex, others rather straightforward but frequently overlooked — that have been used to solve tangible SAS coding problems. In addition, several illustrations of source code debugging are presented as the reader is challenged to derive alternative solutions to each programming puzzle.

**Puzzle 1: Raw Data from Hell**

This puzzle demonstrates the exceptional power of SAS INPUT statements. Sometimes users provide raw data files that have truly bizarre formats. It is possible that these files have been designed by third generation programmers and administrators for 1960's style databases. Or maybe the data entry personnel didn't follow the specified file layout consistently. Several of our most "interesting" experiences will illustrate special features of the INPUT statement that may be needed to handle non-standard cases.

The data originate from a survey which contains a general information section and three sections of subject-specific questions. We want the general information output to a SAS data set called HEADER while the subject-specific responses should be output to data sets SECT1, SECT2 and SECT3. Invalid subject-specific data records are to be output to the data set BAD_DATA.

Following is an example of a raw data file received for input processing and a brief summary of its characteristics. There are multiple input lines per survey respondent, a favorite of individuals who still believe in the card-image record format even though the line is made up of variable-length records.

(D) The first line for each respondent should be ignored. This line contains technical information about the survey form such as an internal mailing code and is of no interest to us.

(B) On the second line, columns 1-6 have a character field right-justified; it needs to be left-justified in the SAS data set. This field is the respondent KEY.

(C) The next four columns (7-10) should be ignored.

(D) Starting somewhere after column 10 is a NAME of varying length but never longer than 32 characters. It may contain individual embedded blanks and is separated from the next field by multiple blanks.

(E) The field after the name is a DATE (in mm/dd/yy form), but the distance between the end of the name and the start of the date varies.

(F) Following the data is a two-character postal code representing the STATE of the respondent's address.

(G) Somewhere after the state code is a string stating the SOURCE of the response in the form SOURCE=xxxxx.

(H) The next line contains an ADDRESS of up to 80 characters. If the address has leading blanks, they must be retained. If the address is shorter than 80 characters, the physical record will be shorter than 80 characters.

(I) On next line is a PERCENTage, left-justified without a decimal point in columns 1-3. This needs to be converted to a fractional value (e.g., 1.00 or 0.60) in the SAS data set.

(J) Starting at column 5 on the same line, there are up to 10 character fields of 3 digits each, representing TYPES of materials and separated from each other by a single blank. It is very possible that fewer than 10 fields will actually be present. If so, the excess fields should be assigned missing values.

All of the above fields (KEY, NAME, DATE, STATE, SOURCE, ADDRESS, PERCENT, TYPE1-TYPE10) should be stored in the SAS data set HEADER.
(K) On the next line(s) are an unknown number of triplicates, each consisting of two character fields (survey SECTION and QUESTION code) and a numeric field (ANSWER), all separated by exactly one blank space. The three fields are never split between two lines.

(L) To signal the end of a response, the first field is present and starts with an asterisk (*) while the other two fields are missing.

(M) The numeric input field can have four possible values: 1—yes or positive, 0—no or negative, .—response not legible, and N—not sure. Unanswered questions will not appear in the input file. All of these four cases must be stored and distinguished from each other in the output SAS data set.

For each valid triplicate, an output record containing KEY, QUESTION and ANSWER should be placed in SAS data set SECT1, SECT2 < SECT3, based on the value of the first field (I, II or III). For any other value of the first field, a record containing KEY, SECTION, QUESTION and ANSWER should be output to the SAS data set BAD_DATA.

How would you handle these data conditions?

Puzzle 2: All the King’s Horses and All the King’s Men

Suppose we want to put together the three subject-specific data sets created in puzzle 1 into a single data set. In order to distinguish between sections, we need to restore the section number that was read in as the first field of the triplicates. This time, however, the section number should be a numeric field (1, 2, or 3) instead of I, II or III.

How would you accomplish this? Would your solution change significantly if there were 40 input data sets (named SECT1, SECT2, ..., SECT40) instead of only three?

Puzzle 3: Doing the "Data Set Twist"

To produce a section profile for each respondent, we need to convert the data from one observation per respondent/question to one observation per respondent with each question as a separate variable. Then, for every respondent, we want to count the number of: (a) positive answers (ANSWER=1); (b) negative answers (ANSWER=0); and (c) all other responses plus unanswered questions. It is known that each section contains questions A through J and the records are sorted by KEY and QUESTION.

A sample of data from the SECT1 data set:

<table>
<thead>
<tr>
<th>KEY</th>
<th>QUESTION</th>
<th>ANSWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>61134</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>61134</td>
<td>C</td>
<td>1</td>
</tr>
<tr>
<td>61134</td>
<td>G</td>
<td>0</td>
</tr>
<tr>
<td>61134</td>
<td>I</td>
<td>0</td>
</tr>
</tbody>
</table>

The initially proposed solution is:

```
PROC TRANSPOSE DATA=SECT1 OUT=SECTION; BY KEY; ID QUESTION; VAR ANSWER; RUN;
```

```
DATA SECT1TOP: SET SECTION;
ARRAY QUESTS (*) A--J;
POSITIVE=SUM(OF A--J);
NEGATIVE=SUM(OF .--N);
NOANSWER=SUM(OF .--N)-POSITIVE-NEGATIVE;
RUN;
```

The results are:

<table>
<thead>
<tr>
<th>KEY</th>
<th>POSITIVE</th>
<th>NEGATIVE</th>
<th>NOANSWER</th>
</tr>
</thead>
<tbody>
<tr>
<td>61134</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>62555</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Why didn’t this solution work? What would you change to make it work? Is there a more straightforward way to calculate a value for NOANSWER?

Puzzle 4: Old Habits Die Hard

After converting the section data in puzzle 3 to the respondent level, the records were summarized to give us the percentage of positive answers for each question, ignoring missing values. The SAS data set SECT1AVG holds the mean value of each question.

```
PROC SUMMARY DATA=SECT1AVG; VAR A--J; OUTPUT OUT=SECT1AVG MEAN=; RUN;
```

```
SECTION
A  B  C  D  E  F  G  H  I  J
0.1 0.2 0.1 0.2 0.3 0.6 0.1 0.1 0.2
```

Now we want to know how many questions had a positive response percentage of more than 50%, that is, greater than 0.5 as a mean value. Although it is clear that only one question (G) satisfies the criterion, the following code produced a result of 2.

```
DATA SECTIONS: SET SECT1AVG;
ARRAY QUESTS (*) A--J;
DO i=1 TO DIM(QUESTS);
   IF QUESTS(i) > 0.5 THEN N=1;
END;
PUT "50% POSITIVE FOR " N " QUESTION(s)";
```

What’s the problem?
Puzzle 5: When Will It End?

We must read a raw data file containing multiple data records on the same line following a unique record key. The eight-character ID occupies positions 1-8 followed by one blank space. Then, any number of repeating groups of data follow, each with a length of 229 bytes. Because there will be a varying number of data groups on each input record, records are variable-length. However, there is a maximum of 10 data groups per record so the maximum record length is 2290.

The fields in each data group are, in order, NAME (character, length 20), STATE (character, length 2), COMMENT (character, length 200) and WEIGHT (numeric, input length 7).

How can we create a SAS data set containing these four variables plus the variable ID?

Puzzle 6: Wasted Space and Wasted Bytes

The SAS data set created in puzzle 5 is projected to contain approximately 1,000,000 observations, requiring about 240 megabytes of storage space. It is discovered that the variable COMMENT is non-blank only about 5% of the time. That translates to roughly 200 megabytes of wasted space. We need to find a way to save space and avoid storing 200 blank characters when the COMMENT variable contains no useful information.

How would you do this?

Puzzle 7: A Slice of the Pie

To test programs that will utilize the input data described in puzzle 5, we have chosen to use a random selection of observations from the large SAS data set. Because the records are assumed to be in no particular order within the SAS data set, we have decided to pick out a "chunk" of data, that is, a group of consecutive observations, which represent a user-specified percentage of the file (typically 5%-10%). When creating the output sample data set, the sparse field split away in puzzle 6, COMMENT, should be restored. In addition, the sample data set should contain not only the 2-character state postal code, but also the corresponding state name in upper- and lowercase.

Can you write the necessary SAS source code?

Puzzle 8: The Chameleon Format

When printing the data described in puzzles 5 through 7, the WEIGHT field causes certain problems. The absolute value of WEIGHT is no larger than 99,999 and no smaller than 1,000,0, but, because the records represent transactions, the actual value may be negative. For each record, the value of WEIGHT must be displayed using an appropriate format based of the size of the number. The format must show 5 significant digits in all cases and insert a comma for absolute values of 10,000 or greater.

Can you create the requisite format using PROC FORMAT?

Solution to Puzzle 1

Each of these points can be handled by a specific feature of the INPUT statement or an associated statement.

(A) Redirecting the input pointer to a particular line, or skipping over lines, can be accomplished with the # line pointer control feature. Specifying "#2" skips over the next line in the input file, positioning the input line pointer on the second line in the remainder of the file.

(B) Using column input for the variable KEY will automatically left-justify the character value.

(C) Skipping the next four columns is done by using the + column pointer control feature and specifying "+4".

(D) Using list input causes the input pointer to search for a non-blank character. Using the $ modifier indicates that NAME can contain individual embedded blanks and the value is terminated as soon as multiple blanks are encountered. To guarantee the maximum length of 32, a LENGTH statement is placed prior to the INPUT statement.

(E) We can read DATE directly using formatted input with the MMDDYYS informat, but since the starting column is unknown, the informat must be preceded by the : format modifier.

(F) Reading STATE is done using list input and the simple $ modifier. The value starts with the first non-blank character.

(G) Using named input will convert the input string SOURCE=xxxxx to a character variable SOURCE with the value xxxxx. The length of the variable SOURCE is established with a LENGTH statement in front of the INPUT statement.

(H) To skip to the next line, use the / line pointer control character. Then, leading blanks in ADDRESS are kept intact by using the $CHAR format. The format is
modified to $CHAR80$ to assign a length of 80 to the ADDRESS variable. Since we are reading a single field, a value less than 80 characters in length will simply be padded with blanks on the right.

(I) After skipping the next line again with the / line pointer control character, PERCENT is read with column input and a ",", decimal specification, implying two decimal places and automatically dividing each value by 100.

(J) To start at column 5 we use the @ column pointer control feature. We can read multiple fields with the same format by specifying an abbreviated variable list (TYPE1-TYPE10) and a grouped format list (@S$.+1$). If we occasionally expect fewer values than variables specified, adding the MISSOVER option to the INFILE statement causes excess variables to be assigned missing values.

(K) Since we do not know how many triplicates will be present, we must allow SAS to read an infinite number of lines. This is done by using 0 (zero) as the termination condition of a "DO UNTIL" loop; since 0 is always false, the loop won't end unless we take further action inside it. (See item L.)

In addition, we must be able to read the same line multiple times if two or more triplicates appear on a single line. This requires use of the "trailing @" line hold specifier to allow multiple reads from the same line. As a result, we must explicitly release any line that has no more data prior to detecting a termination condition. (See item L.)

(L) To detect the end of all triplicates, we must read the first field separately and test it. To check for a value that starts with an asterisk (*), use the ",*" comparison operator. If a termination condition is detected, we must complete processing for the respondent and set up the next respondent to start at the subsequent input line. The RETURN statement serves both purposes since it ends the execution of the DATA step and releases the input line which was just read. If no termination condition is encountered, the other two fields can be read normally.

(M) Introducing the "MISSING N" statement will cause the value "N" in a numeric input field to be read as a special SAS missing value (.,). While the value ",0" will become the standard SAS missing value (.,). No warnings or "INVALID DATA" messages will be produced.

The resulting SAS source code looks something like the following. Note that all of the fields that are output to the HEADER data set are read with a single INPUT statement.

```
DATA HEADER (STATE-STATE DATA STATE SOURCE
ADDRESS TYPE1-TYPE10)
SECT1 (STATE-STATE QUESTION ANSWER)
SECT2 (STATE-STATE QUESTION ANSWER)
SECT3 (STATE-STATE QUESTION ANSWER)
BAD_DATA (STATE-STATE QUESTION ANSWER SECTION)
/
LEN STATE $32 STATE $2 SOURCE $12;
INFILE RANFILE MISSOVER;
INPUT $2 KEY $1-6 +4 NAME $ & DATE :MONDDDD.$;
STATE $ SOURCE=
ADDRESS $CHAR80;
/
PERCENT 1-3 .2 $S(TYPE1-TYPE10) ($3. +1$)
/
OUTPUT HEADER;
MISSING N;
DO UNTIL (0);
INPUT SECT1 $ SECT1; SECT2 $ SECT2; SECT3 $ SECT3;
IF SECT1 "**" THEN RETURN;
ELSE IF SECT1 "*" THEN INPUT;
ELSE DO;
    INPUT (QUESTION ANSWER) ($1.) $;
    SELECT (SECTION);
    WHEN ('Y') OUTPUT SECT1;
    WHEN ('Y') OUTPUT SECT2;
    WHEN ('Y') OUTPUT SECT3;
    OTHERWISE OUTPUT BAD_DATA;
END;
/
END;
RUN;
```

Unfortunately, there is one problem with this code, specifically with respect to the INPUT statement that reads the QUESTION and ANSWER variables. Can you find it?

(solution after the solution to the final puzzle)

Solution to puzzle 2

A straightforward solution is:

```
DATA COMPLETED;
SET SECT1 (£=SECT1);
SECT2 (£=SECT2);
SECT3 (£=SECT3);
IF SECT1 THEN SECT1=1;
ELSE IF SECT2 THEN SECT2=2;
ELSE IF SECT3 THEN SECT3=3;
RUN;
```

Note that all of the IN= variables are automatically dropped from the output data set.

By using the IN= variables as part of an array, we have a more general solution that is easily modified when the number of input data sets changes; the only element that requires modification is the suffix of the final variable in the array. The value of the new SECTION variable is simply the array index that matches the IN= variable of the input data set.
We also could generalize the portion of the code that assigns the section number by using the SAS macro facility. Once again, only the final section number requires modification, although it must be changed in two places.

Solution to Puzzle 3

The main problem is that, despite the fact that the values of QUESTION range from A through J, PROC TRANSPOSE defines variables in the order they are encountered in the input data set. Therefore, the order of variables is A, C, D, J, B, E, and I. Hence, the variable range A-J represents only A, C, D and J.

One way to fix the problem is to define the variables A through J ourselves. Since we need to override the order of variables in the data set SECT1ALL, we must define them prior to the SET statement in the DATA step. This is easily done with a LENGTH statement, "LENGTH A 8 C D E F G H I J 8;", inserted immediately before the SET statement.

The calculation of NOANSWER was specified as the total number of question variables minus the number of 1's minus the number of 0's. The NMISS function can give us this result directly since the only non-zero, non-1 values are missing values. An improved algorithm is "NOANSWER=NMISS(OF A-J);".

Solution to Puzzle 4

The problem results from the fact that the array has been traversed using the loop variable I. The programmer didn't recognize that the variable I is already on the data set; in fact, it is one of the variables being compared to 0.5 in the loop. At the time we've reached the array variable I, the loop variable I is 9. That value of 9 is what is compared to 0.5 and, naturally, it increments the counter inappropriately.

Nested routines can produce similar results. Suppose macros %A and %B are constructed in the following manner:

\begin{verbatim}
ARRAY SECTS [4] SECT1-SECT4;
DO SECTION=1 TO 4;
  IF SECTS(SECTION) THEN DO;
    OUTPUT;
    RETURN;
  END;
END;
\end{verbatim}

Solution to Puzzle 5

The LENGTH option on the INFILE statement can provide the physical length of each input record. We can calculate the number of data groups to be read on each record from this value. We must remember for use the "trailing @@" line hold specifier to maintain access to the same input line when repeatedly reading data groups.

Other points to keep in mind:

- There may be times when the maximum record length must be specified using the LRECL option on the INFILE statement.
- The $CHAR informat will read in a character value exactly as it appears in the input data. Unlike the $w. informat, it does not remove leading blanks and it is capable of reading a single period (.) without interpreting it as a missing value.

Solution to Puzzle 6

A creative solution to this is required because, on most platforms, the SAS System does not allow variable-length character data, although the COMPRESS data set option...
in some Version 6 releases may assist in solving this problem.

Let's create a separate data set (COMMENTS) containing only the COMMENT variable. In the original data set, replace COMMENT with a "pointer" linking to the appropriate observation in the COMMENTS data set. If there is no matching comment, the pointer value is 0 (zero).

DATA GEMINFO (DROP=COMMENT COUNTER)
  COMMENTS (KEEP=COMMENT);
  SET RandData;
  IF COMMENT NE 0:
    DO:
      COUNTER=1;
      COMM_OBS=COMMENT;
      OUTPUT COMMENTS;
    END;
  ELSE COMM_OBS=0;
  OUTPUT GEMINFO;
RUN;

The process of bringing COMMENT values back into the master data set is described in the solution to Puzzle 7.

Solution to Puzzle 7

To select a "chunk" of X percent of data for testing, we must determine the first and last observations to be included. Among the alternative methods of accessing the sample observations are:

1) "Looping" through the entire sequence of chosen observation numbers using the POINT= option of the SET statement

2) Specifying the range of observations with the FIRSTOBS= and OBS= data set options on the SET statement

Our algorithm for choosing X percent of a data set can be summarized as follows: find out the number of observations in the complete data set, select a random number with uniform distribution between 0 and 1; determine the first observation by multiplying the random number by (100-X)% of the total observation count; add X% of the total observation count to get the final record. There are minor adjustments to guarantee valid observation numbers, e.g., minimum value of 1, results rounded to nearest integer.

We use the NOBS= option on the SET statement to get the size of the input data set, but we don't actually read the data set at this point since the value of NOBS is obtained during data step compilation.

We elect to use the RANUNI function, a better random number generator than UNIFORM. To repeat the same sample of observations, we can simply supply the same seed to RANUNI. If we don't supply a seed, the current time (as a SAS time value) will be used.

If we choose alternative method 1, we can read the selected records in the same data step. The variables _1_ and COMM_OBS are automatically dropped since they are used as POINT= variables; likewise, NOBS is automatically dropped. The STOP statement is required since there is no automatic "end-of-data-set" control; we have complete control over the input/output operations in the data step.

For alternative 2, we need to create macro variables using the CALL SYMPUT function in order to pass the calculated observation range to a second data step. Since POINT= tends to introduce considerable processing overhead when reading a large number of records, alternative 2 is often the preferred technique.

We bring back the full value of COMMENT by extracting the appropriate observation from the COMMENTS data set. The 2-character state code can be converted to the corresponding state name in uppercase and lowercase with the built-in SAS data step function STNAMEL; there is no need to create our own table lookup format.

Alternative 1

%MACRO Sample(PCT=10, Seed=100, Xmin());
  DATA SAMPLE (DROP=START STOP);
  LENGTH ST_NAME $ 20;
  START=MIN((Rand(NOB)-1)/100 * RandM(Seed)), 1);
  STOP=MAX((Rand(NOB)-(100/100) * RandM(Seed)));
  DO I = START TO STOP;
    RandGEMINFO POINT=1, NSE=NOBS;
    ST_NAME = STNAME (STDATA);
    IF COMM_OBS > 0
      THEN SET COMMENTS POINT=COMM_OBS;
      ELSE COMMENT='';
    OUTPUT;
  END;
  STOP;
  RUN;
%MEND Sample;

Alternative 2

%MACRO Sample(PCT=10, Seed=100, Xmin());
  DATA SAMPLE (DROP=START STOP);
  LENGTH ST_NAME $ 20;
  IF 0 THEN SET GEMINFO NOBS=NOBS;
    START=MIN((Rand(NOB)-1)/100 * RandM(Seed)), 1);
    STOP=MAX((Rand(NOB)-(100/100) * RandM(Seed)));
    CALL SYMPUT("FIRSTOBS", PUT (START, 12.));
    CALL SYMPUT("LASTOBS", PUT (STOP, 12.));
    STOP;
  RUN;
%MEND Sample;
Solution to Puzzle 8

You can create a customized numeric display format, especially one that changes significantly based on the nature of the data, as a PICTURE format. The following definition satisfies the need for five significant digits based on the assumed data values. If absolute values less than 1.0 were to occur, this format would not be sufficient but could be enhanced to handle ranges between -1 and 1 accordingly.

PROC FORMAT;
    PICTURE SIGSF;
    LOW - -10000 = '99,999' (PREFIX='-')
    -1000 <= -100 = '999,99' (PREFIX='-')
    -100 <= -10 = '99,999' (PREFIX='-')
    -10 <= 0 = '9,9999' (PREFIX='-')
    0 <= 10 = '9,9999'
    10 <= 100 = '99,999'
    100 <= 1000 = '999,99'
    1000 <= 10000 = '9999,9'
    10000 <= HIGH = '99,999'
END;

Correction to Solution to Puzzle 1

The problem is caused by the lack of a comma delimiter between the "$" and the "1." in the grouped format list. Due to the free-form nature of SAS source code, the blank between the intended formats is ignored and the SAS System interprets the grouped format list as a single informat ($1.) which is associated with both variables. As a result, the variable ANSWER is incorrectly created as a character field. This particular problem is addressed under the INPUT statement in several volumes of SAS System documentation.

Acknowledgments

Thanks to the SATO staff of ARC Professional Services Group for contributing their ideas for the puzzles presented in this paper. Special gratitude goes to Sandy Zak, Linda Smith and Bob Hall for assisting in the preparation of the paper itself. Also, we are grateful to our audience at SUGI 15 in Nashville for their positive response to last year’s “Puzzles” presentation, encouraging us to compile this sequel.

References


To contact us

Atlantic Research Corporation
Professional Services Group
1301 Piccard Drive, 2nd floor
Rockville, MD 20850
PHONE: (301) 258-2666
FAX: (301) 258-8878

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