Introduction to the Basics - Avoiding Some Pitfalls

Deb Cassidy, Cardinal Health, Dublin, OH

ABSTRACT

Have you sought help from an experienced user or past SUGI proceedings only to have your job stop running or give you incorrect results after you added the suggestion? For example, using keep/drop options or re-ordering your IF/THEN/ELSE statements can improve efficiency. But if they are used "incorrectly", you can have problems. To add to the confusion, statements that are "incorrect" for one program may be fine for another program. This hands-on workshop will walk you through some basic statements and options that everyone should know. It will focus on showing you the pitfalls that new users (and some experienced users) may encounter and help you to understand how to avoid them.

Topics to be covered will include:
- keep/drop statements and options,
- IF/THEN, IF/THEN/ELSE, and SELECT statements,
- sum function and sum statement,
- creating new variables.

Other topics will be covered in the workshop as time permits. Some of them are included in this paper.

WARNING: SOME OF THE EXAMPLES BELOW SHOW YOU THE WRONG WAY TO DO SOMETHING. PLEASE READ THE ENTIRE SECTION AND NOT JUST THE CODE!!!!

PITFALL EXAMPLE 1
KNOW YOUR DATA

You've been given a SAS® data set containing data and you've been told 'gender' identifies whether the record is for a male or female. Half the data are for each gender. (In our workshop example shown at the end of the paper, we have 5 of each.)

You submit the following code to see the males in the data.

```
DAA MALE;
 SET WKSHP;
 IF GENDER='MALE';
RUN;
```

However, you find only a fraction of the expected observations. What happened? It's a very simple but common problem - you started writing the program before you really knew anything about your data. In this sample data, there were some major data entry inconsistencies. If you aren't familiar with the data you will be using, you should look at it first. Three simple procedures can help.

```
PROC FREQ DATA=WKSHP;
 TABLES _CHARACTER_;
RUN;

PROC MEANS DATA=WKSHP;
RUN;

PROC CONTENTS DATA=WKSHP;
RUN;
```

PROC FREQ and PROC MEANS will show you what your values really are. FREQ should be used for character data and MEANS for numeric. If you don't specify which variables you want, you'll get everything. For MEANS, that is fine because means can only be computed on numeric values. However, you don't want to show the frequencies for a numeric variable with thousands of unique values so you should specify which variables. In the example above, _CHARACTER_ restricts the frequency to just character variables. See PITFALL EXAMPLE 9 for options when you have very large data sets.

PROC CONTENTS will give you more information about the variables. For example, in our workshop data, GENDER was created with a length of 4 which explains why the values for the females were definitely incorrect. (NOTE: If you are using display manager, you can also use the VAR *data-set-name* command to see the same type of information.)

Now that you know you have a data problem, let's assume you want to fix the values. You want to

change the length to 6, capitalize all values and use the full word. You know the LENGTH statement can be used to set the length of a variable and you know the UPCASE function to capitalize values so you try the following code.

```
DATA WKSHP2;
 SET WKSHP;
 LENGTH GENDER $6;
 GENDER=UPCASE(GENDER);
 IF GENDER IN ('FEMA','F','2') THEN
   GENDER='FEMALE';
 IF GENDER IN ('M,'1') THEN
   GENDER='MALE';
RUN;
```

However, you get a WARNING message. Notice that it is not an ERROR message and the job appears to have been successful. The WARNING message tells you the length of the character variable has already been set. It says to use the LENGTH statement as the very first statement in the DATA STEP to declare the length of a character variable. If you look at your new data set, you'll discover you still have problems.

Even though you are creating a new data set WKSHP2, you are starting with the existing data set WKSHP so you can not change the length. You must set the length for the variable in WKSHP2 before you start to read the existing data set as shown below.

```
DATA WKSHP2;
 LENGTH GENDER $6;
 SET WKSHP;
 GENDER=UPCASE(GENDER);
 IF GENDER IN ('FEMA','F','2') THEN
   GENDER='FEMALE';
 IF GENDER IN ('M','1') THEN
   GENDER='MALE';
RUN;
```

PROC CONTENTS will also show you if any of your variables are stored with a format. If they are, the formatted value will appear in your results but you'll need to use the unformatted values in your code. For example, PROC FREQ may show a value of 1998-03-23. However, you'll need the PROC CONTENTS to know if this is stored as a character variable as shown or if it is a SAS date value being printed with a format of yymmdd10. instead of the actual value. The following shows the differences in your code. The latter is the easy way to specify a SAS date. Otherwise, you would have to know the actual value for the number of days since January 1, 1960.

```
IF DATE='1998-03-23';* character;
IF DATE='23MAR98'D;  *SAS date;
```

PITFALL EXAMPLE 2
KEEP/DROP/RENAME OPTIONS

You've been told using KEEP or DROP will make your program more efficient because you aren't carrying along excess variables. You decide to use this idea because you want to create a new data set which only has the IDNUM variable and a new score field which is twice the old score field. You've seen the following examples of KEEP used before in several different ways. Which of the following is appropriate for your purposes?

```
DATA KEEPEX1;
 SET WKSHP2 (KEEP=IDNUM);
 SCORE2=SCORE*2;
RUN;

DATA KEEPEX2;
 SET WKSHP2 ;
 SCORE2=SCORE*2;
 KEEP IDNUM SCORE2;
RUN;

DATA KEEPEX3 (KEEP=IDNUM SCORE2);
 SET WKSHP2;
 SCORE2=SCORE*2;
RUN;


DATA KEEPEX4(KEEP=IDNUM SCORE2);
 SET WKSHP2;
 SCORE2=SCORE*2;
 KEEP IDNUM;
RUN;
```

The first method uses KEEP as an option on an input data set. This means SAS is being told to ignore all the other variables when it reads the data set. Therefore, SCORE doesn't "exist" and you get a NOTE telling you that variable SCORE does not exist. If you look at the new data set, you'll see all missing values for SCORE and SCORE2.

The second and third method are similar. The second uses the KEEP statement while the third uses the KEEP option on the data set. They accomplish the same purpose but the way they are written differ. When you use the data set option, it will be in parentheses and have the = following the keyword KEEP. This isn't true of the KEEP statement.

Both data sets KEEPEX2 and KEEPEX3 will have the correct results. However, you didn't gain much from an efficiency standpoint. The new data sets are more efficient since you don't have any extra variables. However, you had to process all the extra variables to get to that point. If the original data set had thousands of variables, you would have a very inefficient program.

The fourth method doesn't even work. While it is pretty obvious here that you have two contradictory keep statements, it may not be obvious in a longer program. This is one of the reasons that many companies have rules as to whether they want you to use the KEEP statement or the KEEP option on an output data set.

DROP can be used instead of KEEP. Some companies also have very strict rules as to which should be used. The following is legitimate code but NOT recommended since it can be confusing. DATA ONE will end up with only IDNUM because the KEEP option on the output data set has the "final say".

```
DATA KEEPEX5(KEEP=IDNUM);
 SET WKSHP2(KEEP=IDNUM GENDER SCORE);
 DROP GENDER;
RUN;
```

So what is the best way to write the code? From an efficiency standpoint, all of the following will use the least amount of variables for both the input and output data set. From a style standpoint, you should check for your company rules. If you don't have any company rules, you'll want to be consistent within your own work. However, some times the amount of variables you need to type may end up being the deciding factor.

```
DATA KEEPEX6 (KEEP=IDNUM SCORE2);
 SET WKSHP2 (KEEP=IDNUM SCORE);
 SCORE2=SCORE*2;
RUN;

DATA KEEPEX7 ;
 SET WKSHP2 (KEEP=IDNUM SCORE);
 SCORE2=SCORE*2;
 KEEP IDNUM SCORE2;
RUN;

DATA KEEPEX8 ;
 SET WKSHP2 (KEEP=IDNUM SCORE);
 SCORE2=SCORE*2;
 DROP SCORE;
RUN;
```

RENAME can be used as a data set option or statement just like KEEP and DROP. Once again, the main thing to remember is which data set will the RENAME affect.

```
DATA RENAME1;
 SET WKSHP2(KEEP=IDNUM SCORE
     RENAME=(SCORE=NEWNAME));
 SCORE2=NEWNAME*2;
RUN;

DATA RENAME2;
 SET WKSHP2(KEEP=IDNUM SCORE);
 SCORE2=SCORE*2;
 RENAME SCORE=NEWNAME;
RUN;
```

Both of the above examples are correct. In the first case, you are changing the name when you read in the data set so you will use the new name in the processing. In the second case, the RENAME only affects the data set being created so use must use the original name within the data step processing.

PITFALL EXAMPLE 3
IF/THEN/ELSE & SELECT

At first glance, the following code may appear to give you the same values for GROUP_A, GROUP_B, GROUP_C and GROUP_D but two give one answer and two give another.

```
DATA IFTHEN;
 SET WKSHP2;
 IF SCORE<30 THEN GROUP_A=1;
 IF GENDER='MALE' THEN GROUP_A=2;

     IF SCORE<30 THEN GROUP_B=1;
ELSE IF GENDER='MALE' THEN GROUP_B=2;

      IF GENDER='MALE' THEN
GROUP_C=2;
ELSE  IF SCORE<30 THEN GROUP_C=1;

 SELECT;
 WHEN(SCORE<30) GROUP_D=1;
 WHEN (GENDER='MALE') GROUP_D=2;
 OTHERWISE GROUP_D=.;
 END;

RUN;
```

You will get identical answers except for the males who scored under 30. Since these people meet both criteria, where do they fall? For GROUP_A, each IF statement is checked and the GROUP_A value will be

overwritten each time a condition is met. Any change in the order of the statements will affect the results. This method is also very inefficient.

If you use IF/THEN/ELSE statements, SAS will stop checking values after a condition is met. This will be more efficient code. It will also be more efficient if the values most likely to occur in your data are placed first.

However, while GROUP_B and GROUP_C both use IF/THEN/ELSE, you will get different answers because once again the order makes a difference. When your categories are not mutually exclusive, the most efficient code may not give you the right answer.

The SELECT statement is an alternative to IF/THEN/ELSE and is usually more efficient. You should use a select group rather than a series of IF-THEN statements when you have a long series of mutually exclusive conditions. What happens when they are not mutually exclusive? Once again, the efficiency occurs because SAS stops checking after a condition is met so GROUP_D will be assigned the value from the first condition that was met.

Thus, GROUP_A and GROUP_C assign the males scoring under 30 a value of '2' while GROUP_B and GROUP_D will have a value o f'1'.

PITFALL EXAMPLE 4
SUM VS +

Many people will look at this code and say NEXTSCR will equal NEXTSCR2.

```
DATA SUMPLUS;
 SET WKSHP2;
 NEXTSCR=SCORE+1;
 NEXTSCR2=SUM(SCORE,1);
RUN;
```

They will be equal except when SCORE is missing. Adding something to a missing value will give you a missing value. However, if you use the SUM function, you will get the sum of all the non-missing values. Other functions such as MEAN also use the non-missing values. Which one is right? It all depends upon what you are trying to do. In some cases you may want the missing value as the result and in other cases you will not.

PITFALL EXAMPLE 5
CREATING NEW VARIABLES

```
DATA NEWVAR;
 SET WKSHP2;
    IF SCORE<62 THEN RATING='LOW';
ELSE RATING='HIGH';
RUN;

PROC FREQ DATA=NEWVAR;
 TABLES RATING ;
RUN;

PROC CONTENTS DATA=NEWVAR;
RUN;
```

What happens when you run the above code? The frequency will show RATING has values of 'LOW' and 'HIG'. Why? SAS creates the length of a variable based on the first time it is encountered. The first value was "LOW" so SAS assumed it should be a length of 3. You can verify this in the PROC CONTENTS. Had you reversed the code and used

```
     IF SCORE>=62 THEN RATING='HIGH';
ELSE RATING=LOW;
```

you would not have had a problem. However, an even better method is to use a LENGTH statement to set the length as shown.

```
DATA NEWVAR;
 SET WKSHP2;
 LENGTH RATING $4;
     IF SCORE<62 THEN RATING='LOW';
ELSE RATING='HIGH';
RUN;
```

What happens with the following code? The intent of GEN1 is to create a 1-character variable with the first letter of the GENDER variable. The SCAN function will return a given 'word' from a variable with the 'words' being determined by the delimiter specified. (Note that the GENDER variable in our workshop data only has one word and is just used as an example.) The next two statements concatenate two variables.

```
DATA NEWFLD;
 SET WKSHP2;
 GEN1=SUBSTR(GENDER,1,1);
 GENSCAN=SCAN(GENDER,1,',');
 GENSTATE=GENSCAN || STATE;
 STATEGEN=STATE || GENSCAN;
RUN;
```

```
PROC FREQ DATA=NEWFLD;
 TABLES GEN1 GENSCAN
        GENSTATE STATEGEN;
RUN;
PROC CONTENTS DATA=NEWFLD;
RUN;
```

When you look at the frequency, everything appears correct for GEN1 and GENSCAN. However, if you look at the PROC CONTENTS, you'll see some problems. You wanted GEN1 to just be a 1-character variable yet it has a length of 4. How did GENSCAN end up with a length of 200 when you started out with a length of 6? When you create variables with a function, a default length is used. Different functions have different defaults. For example, the default for SUBSTR is the length of the original variable while the default for SCAN is 200. The extra 3 bytes in GEN1 may not seem like a lot but an extra 194 bytes in GENSCAN can cause problems. The moral of the story is to use the LENGTH statement for efficient coding. The length should be the maximum possible length for the new values which means the new variable length can be shorter than the original variable.

Now what happened when you concatenated GENSCAN and STATE? When you put GENSCAN first you ended up with values like 'MALE' instead of 'MALEOH' but when you put STATE first you did get the expected results of 'OHMALE'. Why did one work and not the other? SAS has a maximum length of 200 bytes. Since GENSCAN started with a length of 200, it is really padded to be the full 200 bytes so there wasn't any room to add the characters from STATE. However, STATEGEN isn't really right either. It really contains the 2 characters from STATE and the first 198 characters from GENSCAN. It just happens that you only lost the padded characters.

PITFALL EXAMPLE 6
PARENTHESES

Will TEST1, TEST2, and TEST3 have the same results in the following code?

```
DATA PAREN;
 SET WKSHP2;
 TEST1=SCORE-IDNUM*2;
 TEST2=(SCORE-IDNUM)*2;
 TEST3=SCORE-(IDNUM*2);
RUN;
```

There are some combinations that will give the same value in all three cases but not always. For example, if both SCORE and IDNUM are 0, you'll get 0 in all three cases. However, in most cases, only TEST1 and TEST3 will be equal.

The SAS Language Reference manual provides you a table which shows the priority. Look for OPERATORS in the index. Expressions in parentheses are evaluated first so you should use parentheses to make sure you get what you are expecting.

If IDNUM =1 and SCORE=4 you'll get the following. Since multiplication happens before subtraction, the multiplication in TEST1 will give you a value of 2. The next step is to take SCORE-2 and get a result of 2.

TEST3 will be the same since the parentheses are evaluated first and the multiplication is the expression in the parentheses.

TEST2 will be different because the subtraction is in the parentheses. Evaluating the parentheses will be 4-1 which is 3. The next step is to multiply 3*2 which is 6.

PITFALL EXAMPLE 7
VALUE VERSUS VARIABLE

You've submitted the following code. You don't receive any errors or warning messages yet you don't get any observations. However, when you look at the example data, you definitely see observations where the region is NW.

```
DATA VALUEVAR;
 SET WKSHP2;
 IF REGION=NW;
RUN;
```

If you run the following code, you'll see the problem.

```
DATA VALVAR2
 SET WKSHP2(KEEP=REGION);
 IF REGION=NW;
RUN;
```

You get the note that variable NW is uninitialized. Look again at your original data and you'll see there is a variable name NW as well as the variable REGION having values of NW. When you are referring to character values, you must use quotes. Otherwise, you'll be referring to a variable name. In this example, REGION never has the same value as the

variable NW. However, they could have the same value so you may actually get results but they will be wrong. If one variable is character and the other is numeric you will get an NOTE which should make your problem obvious.

PITFALL EXAMPLE 8
MERGE

Actually, the pitfalls of MERGE have been presented before as an entire workshop. I've include this MERGE example to show you once again that you should really know your data. Some people will look at the following code and not only think it is correct but that both MERGE1 and MERGE2 will be identical.

```
DATA MERGE1;
 MERGE WKSHP2 WHKSHP3;
 BY IDNUM;
RUN;

DATA MERGE2;
 MERGE WKSHP3 WKSHP2;
 BY IDNUM;
RUN;
```

First, the code does use the BY statement which is often forgotten. You also don't get an error messages, warning messages or notes so you may not think there is a problem. However, when you look at the data you'll discover they are different. MERGE1 resulted in a score of 31 for IDNUM 1 but MERGE2 resulted in 21. Yet when you look at IDNUM 2 you see both have the same value. What happened? If the variable is in both data sets and it isn't a BY variable, then the value from the last data set read will over-write the previous value. Since WKSHP3 doesn't have a value for IDNUM 2, it wasn't over-written.

In some cases, you may want the value from the last data set. In other cases, you may want to use the value from a specific data set. If so, just use a KEEP or DROP to avoid the duplicate names. You may want to read in both values and then use some kind of logic to pick which one you want. RENAME will be needed in that case. Here are some examples.

```
DATA MERGE3;
 MERGE WKSHP2(KEEP=IDNUM SCORE)
       WHSHP3(KEEP=IDNUM STATE);
 BY IDNUM;
RUN;

DATA MERGE4;
 MERGE WKSHP2(RENAME=(SCORE=SCORE2))
       WKSHP3(KEEP=IDNUM STATE);
 BY IDNUM;
 IF GENDER='MALE' THEN
      FINSCORE=SCORE;
ELSE IF GENDER='FEMALE' THEN
      FINSCORE=SCORE2;
RUN;
```

Now take a look at STATE which was also in both data sets. In MERGE2, the value from WKSHP2 really did over-write the value from WKSHP3. However, in MERGE1 you ended up with values like "NE" instead of "NEW YORK". "NE" wasn't in either data set. You should notice the pattern that your results are the first 2 characters of the value. If your data sets have different lengths for the same variable, the length will always be the first one encountered. Thus, the longer state name is being truncated to a length of 2. The KEEP/DROP and RENAME options mentioned above can also be used here to solve your problems. You can also specify a LENGTH statement before the MERGE.

A different problem occurs if your data sets happen to have one numeric and one character variable with the same name. You'll get an error message and the job will stop processing. Specifying a LENGTH won't help in this case but the other choices will.

PITFALL EXAMPLE 9
RESTICTING THE NUMBER OF OBSERVATIONS

You may wish to restrict your processing especially when you are getting your code to work. This is especially true if you have a very large data set which millions of observations. Although PITFALL #1 recommends you do a frequency and means on all your data, you may wish to restrict how much you check. You'll need to use your judgment as to whether reducing processing time is worth the potential to overlook a data problem.

You can restrict the number of observations with an option statement as shown where xxx is where you stop processing.

```
OPTIONS OBS=xxx;
```

This statement will be in effect until you issue another OBS= option or until the end of the "job". If you are using display manager, the "job" is really your entire

display manager session so the option will be in effect until you close your session. Simple closing the program editor window and re-opening it won't help.

You can reset the option to be all observations with:

```
OPTIONS OBS=MAX;
```

You can also restrict the number of observations for a particular procedure.

```
PROC FREQ DATA=WKSHP2(OBS=5);
RUN;
```

FIRSTOBS is a similar option where you cause processing to begin with that particular observation. To reset FIRSTOBS, just specify FIRSTOBS=1;

However, there are some potential problems. You can not use OBS or FIRSTOBS when you also restrict your data with a WHERE statement or option. If your FIRSTOBS value exceeds the number of observations, you'll get a warning message and no results.

What if you wanted to see just observations 5, 6, and 7? You might think the following will work.

OPTIONS FIRSTOBS=5 OBS=3;

However, it doesn't. OBS is often thought to represent the number of observations to process. However, it really tells the system the observation number of the last record to process.

If you use FIRSTOBS and OBS together, OBS must be greater than FIRSTOBS. To meet the above requirement you would need this option.

OPTIONS FIRSTOBS=5 OBS=7;

Are you thinking that 5+3 is 8 so shouldn't it be OBS=8? Remember that the first observation itself is counted so you need to add one less than what you want.

SUMMARY

You can do lots of things with SAS. However, you must know your data and know what you want to accomplish to know what is the "right" code. In most of the examples included here, you did not get an ERROR message. In fact, you may not even get a NOTE or a WARNING. What you did wasn't always "wrong" from a SAS language perspective but it may have been wrong from a logic perspective.

In addition to reviewing you log for ERRORS, WARNING and NOTES, you should make sure you review your output closely to see if it is what you expected. You should also make sure that if you have copied code from somewhere else, that you understand what it really does.

WORKSHOP DATA

DATA SET WKSHP1
The variables are IDNUM, GENDER, SCORE, REGION, STATE and NW.

```
1  male   21 NW OH Y
2  MALE   34 SE OH N
3  Male   54 NW NY Y
4  M      76 NE KS N
5  fema   44 SO WV N
6  FEMA   21 NE TN N
7  Fema   22 NE MI N
8  F      17 NE CA N
9  1      84 NE ID N
10 2      23 NE ME N
```

DATA SET WKSHP3
The variables are IDNUM, SCORE, and STATE.

```
1  31 OHIO
3  74 NEW YORK
4  56 KANSAS
5  24 WEST VIRGINIA
6  61 TENNESSEE
7  72 MICHIGAN
8  87 CALIFORNIA
9  34 IDAHO
10 23 MAINE
```

TRADEMARK

REFERENCES

SAS Institute Inc., SAS® Language: Reference, Version 6, First Edition Cary, NC: SAS Institute Inc., 1990. 1042 pp.

CONTACT INFORMATION

Deb Cassidy
Cardinal Health
5555 Glendon Court

Dublin, OH 43016
614-717-7136
dcassidy@cardhealth.com