Alternatives to SAS® IF-THEN/ELSE Processing

Imelda C. Go, Lexington County School District One, Lexington, SC

ABSTRACT

IF-THEN/ELSE statements are simple and easy to use. However, IF-THEN/ELSE statements have their limitations. They are not always easy to read or to make changes to. They may also be less efficient than other methods that are available in SAS. Alternatives discussed include SELECT groups, ARRAY processing, and PROC FORMAT. The discussion includes examples about creating new variables out of existing ones, recoding variable values, validating data, and controlling output appearance.

INTRODUCTION

IF-THEN/ELSE statements are basic statements for conditional processing. They are simple, easy to learn, and easy to use. However, they may not always provide the easiest solution to programming problems. Fortunately, there are a number of alternatives to IF-THEN/ELSE processing.

The paper is in question-and-answer format and will show through contrived examples alternatives to IF-THEN/ELSE processing. The alternatives provided are not necessarily the only or best ways to handle specific programming situations.

Do you use IF-THEN/ELSE statements only for conditional processing?

The following assigns the `teacher` and `counselor` values based on `rating`.

```
data one;
  length teacher counselor $30.;
  input rating $20.;
  if rating='Exemplary'
     then teacher='Frodo';
  else if rating in ('Poor', 'Fair')
     then do; teacher='Aragorn';
          counselor='Gandalf'; end;
  else do; teacher='unassigned';
       counselor='Legolas'; end;
cards;
…
;
```

A SELECT group may be used instead.

```
data one;
  length teacher counselor $30.;
  input rating $20.;
  select (rating);
    when('Exemplary') teacher='Frodo';
    when('Poor','Fair')
          do; teacher='Aragorn';
              counselor='Gandalf'; end;
    otherwise
     do; teacher='unassigned';
     counselor='Legolas'; end;
  end;
cards;
…
;
```

Do you create datasets with subsetting IF-THEN/ELSE statements only to use the resulting data sets as input for exactly the same procedure(s)?

The following creates two data sets: one for males and one for females.

```
data males females;
  input sex $1. grade 2.;
    if sex='M' then output males;
      else if sex='F' then output females;
cards;
…
;

proc freq data=males;
    tables grade;

proc freq data=females;
    tables grade;
```

When the resulting data sets are mutually exclusive subsets of the original data set and they are used with exactly the same procedures, then BY-group processing can be used with procedures that support BY-group processing. In the rewritten code below, the data also has to be sorted according to the variable specified in the BY statement for PROC FREQ.

```
data one;
  input sex $1. grade 2.;
    if sex notin ('M', 'F') then delete;
cards;
…
;

proc sort; by sex;

proc freq; by sex;
    tables grade;
```

Do you create datasets with subsetting IF statements in different DATA steps only to use them as input for exactly the same procedure(s)?

The following creates two data sets: one for 10th grade males and one for 7th grade females.

```
data one;
  input sex $1. grade 2.;
cards;
…
;

data M10;
  set one;
  if sex='M' and grade=10;

proc freq data=M10;
  tables grade;

data F7;
  set one;
  if sex='F' and grade=7;

proc freq data=F7;
  tables grade;
```

When exploratory data analysis is performed, the analyst may need to look at several subsets of data to see if anything of interest might appear. Instead of creating a data set for each subset of interest, use the WHERE statement to specify a subset of the data for the procedure.

```
data one;
  input sex $1. grade 2.;
cards;
…
;

proc freq;
  tables grade;
  where sex='M' and grade=10;

proc freq;
  tables grade;
  where sex='F' and grade=7;
```

There is also the WHERE= data set option.

```
data one;
  input sex $1. grade 2.;
cards;
…
;

proc freq data=one
     (where=(sex='M' and grade=10));
  tables grade;

proc freq data=one
     (where=(sex='F' and grade=7));
  tables grade;
```

Do you create new variables with conditional statements only to control the appearance of output?

In the example below, the `gender2` variable is created for the sole purpose of printing more user-friendly values of M and F (instead of 1 and 2) in PROC FREQ output.

```
data one;
  input gender;
        if gender=1 then gender2='F';
    else if gender=2 then gender2='M';
cards;
…
;
proc freq;
  tables gender2;
```

Instead of creating a new variable, create a user-defined format to control the appearance of output. PROC FREQ will print the values of the `gender` variable as F and M instead of 1and 2.

```
data one;
  input gender;
cards;
…
;

proc format;
  value gender 1='F' 2='M';

proc freq;
  format gender gender.;
```

The `gender.` format may be applied by using the FORMAT statement with a procedure, or it may be applied in the DATA step as shown below. If the format is applied in the DATA step, then the same format will apply to the variable in procedures where the variable is used.

```
proc format;
  value gender 1='F' 2='M';

data one;
  input gender;
  format gender gender.;
cards;
…
;

proc freq;
```

Do you validate data using conditional statements?

Suppose that the valid values for a `gender` variable are 1 and 2 and that other values are invalid.

```
data one;
  input  gender;
  if gender notin (1,2)
     then gender=.;
cards;
…
;
```

An informat can be used to perform simple data validation. If all the valid values for a variable are specified, *all other* values can be considered invalid. The keyword OTHER is used to indicate range values that are not included in all the other ranges for an informat. When _ERROR_ is specified as an informatted value, all values in the corresponding informat range are not valid and a missing value will be assigned to the variable. When _SAME_ is specified as an informatted value, a value in the corresponding informat range stays the same.

Suppose that values from a variable with integer and non-integer values need to be validated and the only valid values are `1` and `2`. The following INVALUE statement uses _SAME_, _ERROR_ , and OTHER for this task:

```
proc format;
invalue check 1=_same_
              2=_same_
             other=_error_;
```

An informat's range can be specified as a list of values separated by commas. The following statement is functionally equivalent to the previous one:

```
proc format;
invalue check 1,2=_same_
              other=_error_;
```

The informat is used in the input statement. SAS will assign a missing value to `gender` if a value other than 1 or 2 is encountered.

```
data one;
  input gender check.;
cards;
…
;
```

Do you create new variables to aggregate data for analysis?

In this example, the user creates the `group` variable to divide the records into four groups based on percentile values. The grouping would determine the values of the tables in the frequencies.

```
data one;
  input  percentile;
       if  1<=percentile<=25 then group=1;
  else if 26<=percentile<=50 then group=2;
  else if 51<=percentile<=75 then group=3;
  else if 76<=percentile<=99 then group=4;
cards;
…
;

proc freq;
  tables group;
```

Instead of creating a new variable for each type of aggregation, use a user-defined format to control the PROC FREQ processing.

```
data one;
  input percentile;
cards;
…
;

proc format;
  value group  1-25=1
              26-50=2
              51-75=3
              76-99=4;

proc freq;
  tables percentile;
  format percentile group.;
```

A format was applied to the `percentile` variable. PROC FREQ will aggregate the data based on the values specified in the format.

When exploratory data analysis is performed, the analyst may need to investigate frequency distributions based on many other groupings. Using formats has its clear advantages in terms of keeping the code readable. Otherwise, many sets of IF-THEN/ELSE statements that define variables with different configurations can potentially clutter the DATA step.

Do you create new variables from existing ones or recode variable values using conditional statements?

For example, the data are letter grades given to students in a class. There is a need to convert the letter grades to numeric grades in order to compute the average grade in that class. The following statements can be added to a DATA step to create a new variable called `numgrade`.

```
       if grade='A'  then numgrade=4;
  else if grade='B+' then numgrade=3.5;
  else if grade='B'  then numgrade=3;
  else if grade='C+' then numgrade=2.5;
  else if grade='C'  then numgrade=2;
  else if grade='D+' then numgrade=1.5;
  else if grade='D'  then numgrade=1;
  else if grade='F'  then numgrade=0;
```

As an alternative, the INPUT function and an informat can be used to achieve the same thing (i.e. create variable `numgrade`) below. The example below also uses the PUT function and the format `$words.` to create variable `text`.

```
proc format;
invalue number
'A'=4         'B+'=3.5
'B'=3         'C+'=2.5
'C'=2         'D+'=1.5
'D'=1         'F'=0;
value $words       'A'='A student'
              'B+','B'='B student'
              'C+','C'='C student'
              'D+','D'='D student'
                   'F'='F student';

data grades;
  input grade $;
  numgrade = input(grade,number.);
  text     = put(grade,$words.);
cards;
A
;
proc print;
```

```
  obs    grade    numgrade      text
   1       A          4       A student
```

The INPUT function returns the value produced when an expression (*source*) is read using a specified informat. The informat type determines whether the result of the INPUT function is numeric or character. The INPUT function is also used to convert character values to numeric values. Its general syntax without optional arguments is:

```
INPUT(source,informat)
```

The PUT function returns a value using a specified format. It writes the values of a numeric or character variable/constant (*source*) using the specified format. The format must be of the same type as the *source*. The PUT function is also used to convert numeric values to character values and always returns a character value. Its general syntax without optional arguments is:

```
PUT(source,format)
```

Do you translate rules found in tables into conditional statements?

Consider the following table that shows a test form number for each grade level.

	Grade 1	Grade 2	Grade 3
Form	47	53	12

The following assigns the value of `form` based on the information in the table above.

```
data one;
  input grade score;
       if grade=1 then form=47;
  else if grade=2 then form=53;
  else if grade=3 then form=12;
cards;
…
;
```

As an alternative, tables can be stored in arrays.

```
data one;
  input grade score;
    array number {3} _temporary_ (47 53 12);
    form=number{grade};
cards;
…
;
```

The alternative uses a temporary array (`number`). Using a temporary array eliminates the need to remove unnecessary variables from the data set had a non-temporary array been involved. Temporary array elements are particularly useful when their values are used for computations. If a temporary array element needs to be retained, it can be assigned to a variable.

At first glance, it looks like there are more statements involved with the alternative. What is the advantage to this method? Think of larger tables. There are tables in one, two, three or even more dimensions. Translating larger tables into IF-THEN/ELSE statements produces a series of statements that doesn't look much like the original table. Many statements are also not very easy to check for errors. Consider the following two-dimensional table that shows the adjusted score based on the age and raw score on a test.

Adjusted Score	Raw Score on a Test				
Age	1	2	3	4	5
13	4	5	5	5	6
14	3	4	5	5	6
15	2	3	4	5	6

The example below shows how an adjusted score can be easily obtained using the ARRAY and without using IF-THEN/ELSE statements.

```
data one;
input age rawscore;
array grid
        {13:15, 1:5} _temporary_
        (4 5 5 5 6
         3 4 5 5 6
         2 3 4 5 6);
adjustedscore=grid(age,rawscore);
cards;
…
;
```

CONCLUSION

As with any programming language, there are a variety of ways to achieve the same task in SAS. Each way has its own advantages and disadvantages. The paper shows that SAS has several versatile and convenient built-in features that serve as alternatives to IF-THEN/ELSE processing. Using the alternatives may result in simplified programming, an economy of code, greater efficiency, and greater readability of programs.

REFERENCES

SAS Institute Inc. (1990), *SAS Programming Tips: A Guide to Efficient SAS Processing*, Cary, NC: SAS Institute, Inc.

SAS Institute Inc., *SAS® Language Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 1256 pp.

SAS Institute Inc., *SAS OnLineDoc®, Version 8*, Cary, NC: SAS Institute Inc., 1999.

TRADEMARK NOTICE

Imelda C. Go (icgo@juno.com)
Lexington County School District One
Lexington, SC