STAT 541

Chapter 13 Supplement: Alternatives to IF-THEN/ELSE Processing

- 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.

IF-THEN/ELSE Statements

- Simple and easy to use.
- Not always easy to read or to make changes to.
- Can be less efficient than other methods.
- Alternatives include SELECT groups, ARRAY processing, and PROC FORMAT. Examples include recoding variable values, validating data, and controlling output appearance.

Conditional Processing

```
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;
…
;
```

SELECT Group

```
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;
…
;
```

Subsetting Conditional Statements

```
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;
```

The two mutually exclusive subsets are used with the same procedure.

BY-Group Processing

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

proc sort; by sex;

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

Subsetting IF Statements

```
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;
```

WHERE Statements

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;
```

WHERE= Data Set Option

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

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

New Variables Just for Output Appearance

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;
```

PROC FORMAT

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.

```
proc format;
   value gender 1='F' 2='M';
data one;
   input gender;
   format gender gender.;
cards;
…
;
proc freq;
```

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 here. 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.

Data Validation

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 not in (1,2)
      then gender=.;
cards;
…
;
```

Data Validation with an Informat

```
proc format;
invalue check 1,2=_same_
              other=_error_;
data one;
  input gender check.;
cards;
…
;
```

Data Validation with an Informat

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

The keyword OTHER indicates range values that are excluded from all the other ranges for an informat. When _ERROR_ is specified as an informatted value, all values in the corresponding informat range are invalid 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.

New Variables for Aggregate Analysis

The user creates the `group` variable to divide the records into four groups based on percentile values.

```
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; var group;
```

User-Defined Format

```
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.;
```

Creating New Variables from Existing Ones

There is a need to convert the letter grades to numeric grades. The following statements 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;
```

INPUT Function and Informats
PUT Function and Formats

```
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';
```

INPUT Function and Informats
PUT Function and Formats

```
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

INPUT Function

The general syntax without optional arguments is:

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

It 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 also converts character values to numeric values.

PUT Function

The general syntax without optional arguments is:

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

It returns a value using a specified format. Writes 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 also converts numeric values to character values and always returns a character value.

Converting Rules in a Table

The table shows a test form number for each grade.

Grade	1	2	3
Form	47	53	12

```
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;
…
;
```

ARRAY Statement

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

Temporary arrays eliminate using unnecessary variables for processing. 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.

Larger Tables

Adjusted Score Depends on the Age and Raw Score

	Raw Score 1	Raw Score 2	Raw Score 3	Raw Score 4	Raw Score 5
Age 13	4	5	5	5	6
Age 14	3	4	5	5	6
Age 15	2	3	4	5	6

ARRAY Statement

The adjusted score is easily obtained using the ARRAY statement 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;
…
;
```