2

Creating Variables Conditionally

2.1 The IF-THEN/ELSE Statement

Creating variables in the DATA step is an essential task in DATA step programming. If the assigned values to the newly created variable are the same across all the observations, the variable can be created by using the assignment statement. However, in most applications, assigned values are often different across the records, and these values are generated depending upon a certain condition. In this situation, a common approach to creating a variable conditionally is to use the IF-THEN/ELSE statement.

2.1.1 Steps for Creating a Variable

When creating a new variable, novice programmers tend to simply create the variables without checking the accuracy of the variables. A proper way to create a variable based on an existing variable should consist of three important steps:

1. Evaluating the existing variable
2. Creating the new variable
3. Checking the accuracy of the newly created variable

Using these three steps is especially important when creating variables conditionally, because assigned values for the newly created variables are not the same across the observations. You have to make sure that each observation acquires its intended value.

Numerous SAS® procedures can be used to evaluate either the existing or the newly created variables. For example, PROC PRINT will give you a general idea what the data looks like. Instead of printing the entire data set, you can use the OBS= data set option to print only the first few observations of the data set. To examine the variable attributes and the total number of observations, you can use PROC CONTENTS. You can use PROC MEANS to evaluate the distribution of numeric variables; using the NMISS option in PROC MEANS is especially useful for examining missing numerical values.

To examine the frequency of a categorical variable, one often uses PROC FREQ. You can use the MISSING or MISSPRINT options in PROC FREQ to check whether a categorical variable contains missing values.

Suppose that you would like to create an indicator variable (AGE_HI) based on the numeric AGE variable in the HEARING data set created in Chapter 1. You would like to assign a value of 1 to AGE_HI when AGE is greater than its median value; otherwise AGE_HI is set to zero (0).

To follow the three-step procedure above, you need to examine the AGE variable first. In addition to knowing the median value of the AGE variable, it is important to know the number of missing and non-missing values for the AGE variable because if the AGE variable is missing for an observation, AGE_HI should be assigned with a missing value as well. All this information can be found from PROC MEANS by using the MEDIAN (median value), N (number of non-missing values), and NMISS (number of missing values) options.

Program 2.1:

```
title 'Evaluate the AGE variable';
proc means data = hearing n nmiss median maxdec = 2;
     var age;
run;
```

Output from Program 2.1:

```
Evaluate the AGE variable
The MEANS Procedure
Analysis Variable : Age
                N
         N     Miss           Median
        ---------------------------
        33        1            26.00
        ---------------------------
```

The output from PROC MEANS from Program 2.1 shows that the median age is 26. There are 33 non-missing values and one missing value for AGE. In the second step (Program 2.2), you can create the variable AGE_HI by using the IF-THEN/ELSE statement, which was introduced in Chapter 1. The syntax is also listed below:

```
IF expression THEN statement;
<ELSE statement;>
```

Program 2.2:

```
data hearing2_1;
    set hearing;
```

```
    if age > 26 then age_hi = 1;
    else age_hi = 0;
run;
```

Once the AGE_HI variable is created, the final (and most important) step is to validate that the newly-created variable (AGE_HI) is indeed correctly created. To check the accuracy of AGE_HI, you need to make sure that the maximum value for AGE within the low AGE_HI group (AGE_HI = 0) is 26 and the minimum value of AGE within the high AGE_HI group (AGE_HI = 1) is greater than 26. Program 2.3 uses PROC MEANS to examine the minimum and the maximum values of AGE by each category of AGE_HI. Furthermore, Program 2.3 also examines the number of missing and non-missing values within each level of AGE_HI by using the N and NMISS options.

Program 2.3:

```
title 'Checking AGE_HI is created correctly';
proc means data = hearing2_1 n nmiss min max maxdec = 2;
    class age_hi;
    var age;
run;
```

Output from Program 2.3:

```
                Checking AGE_HI is created correctly
                        The MEANS Procedure
                      Analysis Variable : Age
                   N                N
    age_hi      Obs       N      Miss      Minimum      Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         0       18      17         1        15.00        26.00
         1       16      16         0        28.00        36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Based on the output from Program 2.3, the AGE range is correct when AGE_HI equals 1. There is a problem, however, when AGE_HI is set to zero in the IF-THEN/ELSE statement. Since missing (.) is a number, in fact the smallest numeric value in SAS, the *missing* value for AGE incorrectly translates to a *nonmissing* zero in AGE_HI.

2.1.2 Handling Missing Values When Creating Variables

In most situations, missing values in the source variable should translate into missing values for the target variable. In the previous example involving AGE_HI, the IF-THEN/ELSE statement should have excluded missing values for AGE.

One way to examine the observations that may contain numerical missing values, which are denoted with periods (.), is to check variables with the EQ comparison operator. For example,

```
if AGE EQ . then... ;
```

If you want to compare a character variable with a missing value, you need to compare the character variable with a blank space enclosed in either single or double quotation marks. Alternatively, you can use the MISSING function to check whether its argument contains any missing values. The MISSING function has the following form:

```
MISSING(numeric-expression | character-expression)
```

The advantage of using the MISSING function is that you don't need to know whether its argument is character or numeric. The argument in the MISSING function can be constant, variable, or an expression. If the argument contains a missing value, the MISSING function will return a value of 1; otherwise, it will return 0. Thus, you can create the AGE_HI variable by excluding observations with missing AGE observations by writing the following statement:

```
if missing(age) eq 0 then
   if age > 26 then age_hi = 1; else age_hi = 0;
```

In the SAS statement above, the IF-THEN/ELSE statement that created the AGE_HI variable is nested within the THEN clause of the *outer* IF-THEN statement. The observations for the AGE_HI variable are assigned with either a 1 or 0 only for observations where AGE is not missing. Program 2.4 correctly creates the AGE_HI variable.

Program 2.4:

```
data hearing2_2;
    set hearing;
    if missing(age) eq 0 then
        if age > 26 then age_hi = 1; else age_hi = 0;
run;

title 'Creating AGE_HI considering the missing value';
proc means data = hearing2_2 n nmiss min max maxdec = 2;
    class age_hi;
    var age;
run;
```

Output from Program 2.4:

```
       Creating AGE_HI considering the missing value
       The MEANS Procedure
```

```
                    Analysis Variable : Age
                      N                N
        age_hi      Obs      N     Miss     Minimum     Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
             0       17     17        0       15.00       26.00
             1       16     16        0       28.00       36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Notice that PROC MEANS in Program 2.4 correctly shows that there are no missing values in both the 0 and 1 strata for the AGE_HI variable. However, the person with the missing AGE was not shown in the output. In order to show the missing values that have been excluded from the output, you need to use the MISSING option in the PROC MEANS statement (see Program 2.5). The MISSING option will consider missing values as valid values to create the combinations of class variables.

Program 2.5:

```
title 'Use the MISSING option to show missing values';
proc means data = hearing2_2 n nmiss min max maxdec = 2 missing;
    class age_hi;
    var age;
run;
```

Output from Program 2.5:

```
         Use the MISSING option to show missing values
                      The MEANS Procedure
                    Analysis Variable : Age
                      N                N
        age_hi      Obs      N     Miss     Minimum     Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
             .        1      0        1           .           .
             0       17     17        0       15.00       26.00
             1       16     16        0       28.00       36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2.1.3 TRUE and FALSE: Logical Expressions

The *expression* in the IF-THEN/ELSE statement often contains a comparison operator, such as EQ, to perform a comparison. The execution of the THEN or the ELSE clause depends upon the value that results from the comparison. In the last example, the value from the MISSING function is compared with 0 by using the EQ operator. If the comparison is evaluated to be TRUE, then the value for the AGE_HI variable is generated. Alternatively, you can

compare the value from the MISSING function with 1 by adding the NOT operator. For example,

```
if not (missing(age) eq 1) then
    if age > 26 then age_hi = 1; else age_hi = 0;
```

In SAS, any numerical value other than 0 or the missing value is considered TRUE, and the missing value and 0 are considered FALSE. Since the MISSING function returns a numerical value with either 1 (for TRUE) or 0 (for FALSE), it is redundant to compare values from the MISSING function with 1 explicitly. You can simply write the following:

```
if not missing(age) then
    if age > 26 then age_hi = 1; else age_hi = 0;
```

In some applications, you also need to compare a numeric variable explicitly with 1. For example, in the HEARING data set, the variable PREG is coded with 1 for indicating being pregnant and 0 for not being pregnant. Suppose that you would like to assign pregnant women to group "A" and nonpregnant women to group "B".

Program 2.6 creates the GROUP variable that is based on the PREG variable. All three steps of creating variables are included in the program. During the first step, the variable PREG is evaluated with PROC FREQ instead of PROC MEANS. PROC FREQ is preferred because PREG has just two distinct values (0 and 1), and frequencies need to be examined at each level.

In the DATA step, the statement "IF PREG THEN..." is equivalent to "IF PREG = 1 THEN...". Once the GROUP variable is created from the DATA step, PROC FREQ returns as a third step to check if GROUP is correctly created by examining the frequency in the cross-tabulation of the PREG and GROUP variables.

Program 2.6:

```
/* Step # 1 */
title 'Check PREG variable';
proc freq data = hearing;
    tables preg/missing nocol nopercent;
run;

/* Step # 2 */
data hearing2_3;
    set hearing;
    if not missing(preg) then
        if preg then group = "A";
        else group = "B";
run;

/* Step # 3 */
title 'Check TRIAL is created correctly';
```

```
proc freq data = hearing2_3;
    tables preg*group/missing norow nocol nopercent;
run;
```

Output from Program 2.6:

```
                Check PREG variable
                 The FREQ Procedure
                 Preg    Frequency
                 — — — — — — — — —
                    .            4
                    0           19
                    1           11

   Check TRIAL is created correctly
          The FREQ Procedure
        Table of Preg by group
  Preg      group
Frequency,       |A     |B      | Total
— — — — -+— — — +— — — +— — — +
       . |    4 |    0 |     0 |     4
— — — — -+— — — +— — — +— — — +
       0 |    0 |    0 |    19 |    19
— — — — -+— — — +— — — +— — — +
       1 |    0 |   11 |     0 |    11
— — — — -+— — — +— — — +— — — +
 Total         4      11      19      34
```

A classic logic error often occurs when using the OR operator in the IF-THEN/ELSE statement. For example, "if foo = 10 or 20" always resolves to TRUE because 20 is treated as a single expression and evaluated as TRUE (you actually intended to write "if foo = 10 or foo = 20"). This type of error is often difficult to detect because the syntax is completely correct.

2.1.4 The LENGTH Attribute

In the previous example, values for GROUP have the same one-character length ("A" or "B"). When creating a character variable that contains values with different lengths, you should preset the variable length before creating the variables; otherwise, the values with longer lengths might be truncated. For example, suppose that you want to create a character variable (AGE_CAT) based on the AGE variable in the HEARING data set.

You would like to assign AGE_CAT to "old" when AGE is greater than its median and "young" when AGE is less than the median. At first glance, you might think that the code in Program 2.7 would work.

Program 2.7:

```
data hearing2_4;
    set hearing;
```

```
    if not missing(age) then
        if age > 26 then age_cat = "old";
        else age_cat = "young";
run;

title 'The first 5 observations of HEARING2_4 data set';
proc print data = hearing2_4(obs = 5);
    var age age_cat;
run;
```

Output from Program 2.7:

```
      The first 5 observations of HEARING2_4 data set
            Obs    Age    age_cat
              1     26      you
              2     26      you
              3     32      old
              4     32      old
              5     34      old
```

Notice that the value "young" is truncated to "you." The reason for the truncation is that when creating character variables, SAS will allocate the same number of bytes of storage space as there are characters in the first value that it encounters in the data set for that variable. In the program above, SAS allocates 3 bytes because "low" was encountered first. To avoid mistakes caused by truncation, *always* use the LENGTH statement described below to define character variables in a data step:

```
LENGTH variable(s) <$> length;
```

You can specify one or more *variables* in a LENGTH statement. The dollar sign ($) in the LENGTH statement is used to apply character variables. In the LENGTH statement, *length* is the number of bytes that are used for storing variable values. The length for numeric variables is up to 8 bytes, and the length for character variables is up to 32,767 bytes. The LENGTH statement must be placed before any other reference to the variable in the DATA step; otherwise it won't take effect. Program 2.8 correctly creates the AGE_CAT variable by placing the LENGTH statement before the IF-THEN statement.

Program 2.8:

```
data hearing2_5;
    length age_cat $ 5;
    set hearing;
    if not missing(age) then
        if age > 26 then age_cat = "old";
        else age_cat = "young";
run;
```

```
title 'The first 5 observations of HEARING2_5 data set';
proc print data = hearing2_5(obs = 5);
    var age age_cat;
run;
```

Output from Program 2.8:

```
The first 5 observations of HEARING2_5 data set
       Obs    Age    age_cat
         1     26     young
         2     26     young
         3     32     old
         4     32     old
         5     34     old
```

2.1.5 DO Group

Sometimes you might find it necessary to execute a group of statements as one unit, which can be accomplished by using the DO statement. The DO statement has the following form:

```
DO;
       SAS statement1
...
       SAS statementn
END;
```

The SAS statements between the DO and END statements are called a DO group. You can nest DO groups within DO groups. A DO group is often used within IF-THEN/ELSE statements. Suppose that you would like to create two variables: PREG_INFO and PREG_SMOKER. These two variables are created by the following conditions:

- PREG_INFO is assigned to
  - 1 if the PREG variable is not missing
  - 0 if PREG is missing
- PREG_SMOKER is assigned to
  - 1 if PREG is not missing and the SMOKE variable equals "current"
  - 0 if PREG is not missing and SMOKE is not "current"

Since both the PREG_INFO and PREG_SMOKER variables require a check to see if PREG is missing, you can utilize a DO group with an IF-THEN/ELSE statement to create these two variables.

Program 2.9 starts with the FREQ procedure before creating the variables PREG_INFO and PREG_SMOKER. Based on the output, you can see that

there is only one pregnant smoker and four people have missing values for the PREG variable. When creating PREG_INFO and PREG_SMOKER in the DATA step, a DO group is used to assign PREG to 1 and to create the PREG_SMOKER variable for the condition when PREG is not missing. The final step uses PROC FREQ again to verify whether these two variables are created correctly.

Program 2.9:

```
title 'Frequency Tables: Preg by Smoke';
proc freq data = hearing;
    tables preg*smoke/missing norow nocol nopercent;
run;

data hearing2_6;
    set hearing;
    if not missing(preg) then
    do;
       preg_info = 1;
       if smoke = "current" and preg = 1 then preg_smoker = 1;
       else preg_smoker = 0;
    end;
    else preg_info = 0;
run;

title 'Check if PREG_SMOKER and PREG_INFO are created cor-
rectly';
proc freq data = hearing2_6;
    tables preg_smoker preg_info/missprint;
run;
```

Output from Program 2.9:

Frequency Tables: Preg by Smoke

The FREQ Procedure
Table of Preg by smoke

Preg Frequency	smoke	current	never	past	Total
.	0	1	1	2	4
0	0	6	9	4	19
1	1	1	8	1	11
Total	1	8	18	7	34

```
   Check if PREG_SMOKER and PREG_INFO are created correctly

                    The FREQ Procedure
                                                 Cumulative  Cumulative
   preg_smoker    Frequency     Percent     Frequency      Percent
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
             .             4           .              .            .
             0            29       96.67             29        96.67
             1             1        3.33             30       100.00
                    Frequency Missing = 4

                                                 Cumulative  Cumulative
   preg_info    Frequency     Percent     Frequency      Percent
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
             0             4       11.76              4        11.76
             1            30       88.24             34       100.00
```

2.2 Executing One of Several Statements

The IF-THEN/ELSE statement in the previous section contains only one optional ELSE clause. With only one ELSE statement, you can execute statements based on two conditions. In the situation where you would like to execute multiple statements that are based on more than two conditions, you can either add multiple ELSE IF clauses to the IF-THEN/ELSE statement or use the SELECT statement.

2.2.1 Multiple IF-THEN/ELSE Statements

Multiple IF-THEN/ELSE statements have the following form:

```
IF expression THEN statement;
ELSE IF expression THEN statement;
<...
ELSE IF expression THEN statement;
<ELSE statement;>>
```

The last optional ELSE clause becomes a default that is automatically executed for all observations failing to satisfy any of the previous IF statements.

Suppose you would like to create a variable (AGEGROUP) based on the AGE variable in the HEARING data set. The AGEGROUP variable will be assigned with values 1, 2, or 3 based on the following conditions:

- AGEGROUP is assigned to 1 for AGE ≤ 20
- AGEGROUP is assigned to 2 for 20 < AGE ≤ 30
- AGEGROUP is assigned to 3 for AGE > 30

Program 2.10 creates AGEGROUP in two different ways. In the first method, AGEGROUP1 is created by comparing AGE with 30, 20, and the missing value (in descending order) with the > comparison operator. When AGE has a value greater than 30, AGEGROUP1 is set to 3, and the remaining conditions are not evaluated. When AGE is greater than 20 but less than or equal to 30, the AGE > 30 condition is checked again but this time it fails, so processing moves on to the next check. This time there is a hit, because AGE > 20 is satisfied. Now AGEGROUP1 is set to 2. Finally, when AGE is less than or equal to 20, all three conditions have to be checked before a value of 1 can be assigned to AGEGROUP1.

When writing multiple IF-THEN/ELSE statements, it is efficient to compare the threshold value in either ascending or descending order. When comparing the threshold values in descending order, either the > or >= operators should be used. On the other hand, when comparing the threshold values in ascending order, you should use either the < or < = operators. In the second method, AGEGROUP2 is created by comparing AGE with 20 and 30 (in ascending order) with the <= comparison operator after excluding the missing values.

Program 2.10:

```
data hearing2_7;
    set hearing;

    *method1;
    if age > 30 then agegroup1 = 3;
    else if age > 20 then agegroup1 = 2;
    else if age >. then agegroup1 = 1;

    *method2;
    if not missing(age) then
        if age <= 20 then agegroup2 = 1;
        else if age <= 30 then agegroup2 = 2;
        else agegroup2 = 3;
run;

title 'Check AGEGROUP1 is created correctly';
proc means data = hearing2_7 missing n nmiss min max maxdec = 2;
    class agegroup1;
    var age;
run;

title 'Check AGEGROUP2 is created correctly';
proc means data = hearing2_7 missing n nmiss min max maxdec = 2;
    class agegroup2;
    var age;
run;
```

Output from Program 2.10:

```
            Check AGEGROUP1 is created correctly
                    The MEANS Procedure
                  Analysis Variable : Age
                    N                N
  agegroup1      Obs        N      Miss      Minimum       Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          .        1        0         1            .             .
          1       10       10         0        15.00         20.00
          2       12       12         0        23.00         30.00
          3       11       11         0        31.00         36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

            Check AGEGROUP2 is created correctly
                    The MEANS Procedure
                  Analysis Variable : Age
                    N                N
  agegroup2      Obs        N      Miss      Minimum       Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          .        1        0         1            .             .
          1       10       10         0        15.00         20.00
          2       12       12         0        23.00         30.00
          3       11       11         0        31.00         36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Program 2.11 illustrates the use of multiple IF-THEN/ELSE statements with the DO group. This program creates two variables, TRIAL and REQUIREINFO, based on the variable PREG.

Program 2.11:

```
data hearing2_8;
    set hearing;
    length trial $4;
    if preg = 1 then do;
        trial = "A";
        requireInfo = 0;
    end;
    else if preg = 0 then do;
        trial = "B";
        requireInfo = 0;
    end;
    else do;
        trial = "Wait";
        requireInfo = 1;
    end;
run;
```

```
title 'Checking if TRIAL and REQUIREINFO are created correctly';
proc freq data = hearing2_8;
    tables (trial requireInfo)*preg/missing nocol norow
      nopercent;
run;
```

Output from Program 2.11:

```
    Checking if TRIAL and REQUIREINFO are created correctly
                    The FREQ Procedure
                  Table of trial by Preg
        trial      Preg
        Frequency|       .|       0|       1|    Total
        - - - - -+- - - - +- - - - +- - - - +
        A        |       0 |       0 |      11 |       11
        - - - - -+- - - - +- - - - +- - - - +
        B        |       0 |      19 |       0 |       19
        - - - - -+- - - - +- - - - +- - - - +
        Wait     |       4 |       0 |       0 |        4
        - - - - -+- - - - +- - - - +- - - - +
        Total            4        19        11         34

               Table of requireInfo by Preg
         requireInfo      Preg
         Frequency|       .|       0|       1|   Total
         - - - - -+- - - - +- - - - +- - - - +
                0 |       0 |      19 |      11 |       30
         - - - - -+- - - - +- - - - +- - - - +
                1 |       4 |       0 |       0 |        4
         - - - - -+- - - - +- - - - +- - - - +
         Total            4        19        11         34
```

2.2.2 Executing Statements Using the SELECT Group

The SELECT group is an alternative method to executing one of several statements, which has the following form:

```
SELECT <(select-expression)>;
     WHEN-1 (when-expression-1 <..., when-expression-n>)
     statement;
     <... WHEN-n (when-expression-1 <..., when-expression-n>)
     statement;>
     <OTHERWISE statement;>
END;
```

A SELECT group consists of several statements that start with the SELECT statement and end with the END statement. In the SELECT statement, the optional *select-expression* is used to specify any SAS expression that can be

evaluated into a single value. Within the SELECT group, it contains at least one WHEN statement, which is used to identify which *statement* is to be executed when a WHEN-condition is met. This WHEN-condition, which is either TRUE or FALSE, depends upon whether the *select-expression* is specified.

When a *select-expression* is specified in the SELECT statement, SAS compares the evaluated results from *select-expression* and *when-expression* and returns a value of TRUE or FALSE. If the comparison is true for a WHEN statement, the corresponding *statement* is executed; otherwise, a comparison is performed for either the next *when-expression* within the current WHEN statement or the one in the next WHEN statement. If there is no WHEN-condition that is TRUE, the OTHERWISE statement is executed if one exists. If all the comparisons are false and there is no OTHERWISE statement, SAS will issue an error message and terminate DATA step execution. If the comparison is TRUE for more than one WHEN statement, only the first WHEN statement is executed.

The execution sequence of the WHEN statement when no *select-expression* is specified is similar to when a *select-expression* is specified. However, when no *select-expression* is specified, only the *when-expression* is evaluated and generates a value of TRUE or FALSE. If it is TRUE for a WHEN statement, the corresponding *statement* is executed.

Executing statements that utilize the SELECT group can also be rewritten by using the multiple IF-THEN/ELSE statements. Using the SELECT group is generally more suitable than IF-THEN/ELSE for statements having multiple conditions that need to be processed.

Program 2.12 uses select groups to create a number of variables. The first SELECT group creates the ETHNIC variable by comparing the values from the RACE variable with two groups of values. If the value from the RACE variable equals either "W" or "H", then ETHNIC will be assigned with the "white" value; if RACE equals "B" or "A", then ETHNIC will be assigned with the "non-white" value. The OTHERWISE statement is not utilized, which is not a recommended practice because if the result of all SELECT-WHEN comparisons is false, SAS will issue an error message. A final FALSE would occur, for example, if RACE is missing or if lowercase characters ("w","h","b","a") are assigned to the variable.

The second SELECT group creates the TRIAL and DRUG variables. The DO group is used to group two assignment statements for each WHEN statement. The OTHERWISE statement is used without using an additional statement after the keyword OTHERWISE. This means that if PREG is other than 1 or 0, the TRIAL and DRUG variables will be assigned missing values.

The third SELECT group creates the GROUP variable. An OTHERWISE statement is used within the SELECT group. Subjects with HEARING other than "yes" and "no" values will be assigned to 3 for the GROUP variable.

The last SELECT group creates the variable HIGHINCOME. In this SELECT group, no *select-expression* is specified. In this situation, the *when-expression* is evaluated for selecting which WHEN statement is to be executed.

Program 2.12:

```
data hearing2_9;
    set hearing;
    length ethnic $ 10;

    select (race);
        when ("W", "H") ethnic = "white";
        when ("B", "A") ethnic = "non-white";
    end;

    select (preg);
        when (1) do;
            trial = "A";
            drug = "Treatment";
        end;
        when (0) do;
            trial = "B";
            drug = "placebo";
        end;
        otherwise;
    end;

    select(hearing);
        when ("yes") group = 1;
        when ("no") group = 2;
        otherwise group = 3;
    end;

    select;
        when (income > 100000) highincome = 1;
        when (income >.) highincome = 0;
        otherwise;
    end;
run;

title 'Check if ETHNIC, TRIAL, DRUG, and GROUP are created
correctly';
proc freq data = hearing2_9;
    tables race*ethnic
           preg*trial
           preg*drug
           hearing*group/norow nocol nopercent missing;
run;
```

```
title 'Check if HIGHINCOME is created correctly';
proc means data = hearing2_9 missing n nmiss min max maxdec = 2;
    class highincome;
    var income;
run;
```

Output from Program 2.12:

```
Check if ETHNIC, TRIAL, DRUG, and GROUP are created correctly
                    The FREQ Procedure
                 Table of race by ethnic
          race     ethnic
          Frequency|non-white|white |Total
          — — — — -+— — — — —+ — —  +
          A        |       5 |    0 |    5
          — — — — -+— — — — —+— — — +
          B        |       5 |    0 |    5
          — — — — -+— — — — —+— — — +
          H        |       0 |    4 |    4
          — — — — -+— — — — —+— — — +
          W        |       0 |   20 |   20
          — — — — -+— — — — —+— — — +
          Total           10     24     34

                  Table of Preg by trial
        Preg      trial
        Frequency|        |A       |B       | Total
        — — — — -+— — — — +— — — — +— — — — +
               . |      4 |      0 |      0 |     4
        — — — — -+— — — — +— — — — +— — — — +
               0 |      0 |      0 |     19 |    19
        — — — — -+— — — — +— — — — +— — — — +
               1 |      0 |     11 |      0 |    11
        — — — — -+— — — — +— — — — +— — — — +
        Total           4       11       19      34

                  Table of Preg by drug
        Preg      drug
        Frequency|        |Treatment |placebo | Total
        — — — — -+— — — — +— — — — —+— — — — +
               . |      4 |        0 |      0 |     4
        — — — — -+— — — — +— — — — —+— — — — +
               0 |      0 |        0 |     19 |    19
        — — — — -+— — — — +— — — — —+— — — — +
               1 |      0 |       11 |      0 |    11
        — — — — -+— — — — +— — — — —+— — — — +
        Total           4         11       19      34
```

```
Check if ETHNIC, TRIAL, DRUG, and GROUP are created correctly
                    The FREQ Procedure
                Table of Hearing by group
          Hearing     group
          Frequency|       1|       2|       3| Total
          - - - - -+- - - - +- - - - +- - - - + 
                   |      0 |      0 |     23 |    23
          - - - - -+- - - - +- - - - +- - - - + 
          no       |      0 |      8 |      0 |     8
          - - - - -+- - - - +- - - - +- - - - + 
          yes      |      3 |      0 |      0 |     3
          - - - - -+- - - - +- - - - +- - - - + 
          Total           3        8       23       34

          Check if HIGHINCOME is created correctly
                    The MEANS Procedure
                Analysis Variable : Income
                  N             N
   highincome    Obs     N    Miss       Minimum       Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            0     31    31       0      13900.00      98000.00
            1      3     3       0     113000.00     134000.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

2.3 Modifying the IF-THEN/ELSE Statement with the Assignment Statement

When creating an indicator variable (a variable with values of 1 or 0), you can use the IF-THEN/ELSE statement. For example,

```
if age>26 then age_hi = 1;
else age_hi = 0;
```

In this situation, you can modify the IF-THEN/ELSE statement by using the *assignment* statement, which was introduced in Chapter 1. The assignment statement has the following form:

```
variable=expression;
```

In the assignment statement, *variable* is either a new or existing variable, and *expression* is any valid SAS expression. The IF-THEN/ELSE statement above can be rewritten by utilizing an assignment statement. For example,

```
age_hi = age>26;
```

The expression age>26 is evaluated with 1 for observations with ages greater than 26, and the value 1 will then be assigned to AGE_HI. For

observations with ages less than or equal to 26, AGE_HI will be assigned to 0. Similarly, you can also use the assignment statement for creating a variable with ordinal numeric values. For instance, in a previous example, AGEGROUP was created by utilizing the multiple IF-THEN/ELSE statement:

```
if age>30 then agegroup = 3;
else if age>20 then agegroup = 2;
else if age>. then agegroup = 1;
```

The multiple IF-THEN/ELSE statement is equivalent to writing the following assignment statement:

```
agegroup = (age>.) + (age>20) + (age>30);
```

The three parentheses in the assignment statement above are necessary to ensure the three comparisons are evaluated before the addition operation because the addition operator has higher evaluation priority than the comparison operators. Examples for creating the AGEGROUP variable by using different age values are summarized in Table 2.1. The variable AGEGROUP is created in Program 2.13.

Program 2.13:

```
data hearing2_10;
    set hearing;
    agegroup = (age>.) + (age>20) + (age>30);
run;

title 'Check if AGEGROUP is created correctly';
proc means data = hearing2_10 n nmiss min max maxdec = 2
  missing;
    class agegroup;
    var age;
run;
```

Output from Program 2.13:

```
            Check if AGEGROUP is created correctly
                      The MEANS Procedure
                    Analysis Variable : Age
                     N              N
   agegroup      Obs      N     Miss     Minimum      Maximum
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
          0        1      0        1           .            .
          1       10     10        0       15.00        20.00
          2       12     12        0       23.00        30.00
          3       11     11        0       31.00        36.00
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

TABLE 2.1

Examples for Creating the AGEGROUP Variable

When AGE Equals...	`(age>.) + (age>20) + (age>30)` Evaluates to the Following	`agegroup` Is Assigned with
. (missing value)	0 + 0 + 0	0
15	1 + 0 + 0	1
25	1 + 1 + 0	2
35	1 + 1 + 1	3

Exercises

Exercise 2.1. Using the IF-THEN/ELSE statement, create the following three variables based on the variables in the GRADE.SAS7BDAT data set:

1. MATH_POINT:
   - MATH_POINT is assigned to 4 if variable MATH = "A"
   - MATH_POINT is assigned to 3 if variable MATH = "B"
   - MATH_POINT is assigned to 2 if variable MATH = "C"
   - MATH_POINT is assigned to 1 if variable MATH = "D"
   - MATH_POINT is assigned to 0 if variable MATH = "F"
2. ENGLISH_POINT:
   - ENGLISH_POINT is assigned to 4 if variable ENGLISH ≥ 90
   - ENGLISH_POINT is assigned to 3 if variable 80 ≤ ENGLISH < 90
   - ENGLISH_POINT is assigned to 2 if variable 70 ≤ ENGLISH < 80
   - ENGLISH_POINT is assigned to 1 if variable 60 ≤ ENGLISH < 70
   - ENGLISH_POINT is assigned to 0 if variable ENGLISH < 60 and not missing
3. PE_GRADE:
   - PE_GRADE is assigned to "pass" if variable PE = 1
   - PE_GRADE is assigned to "no pass" if variable PE = 0

Exercise 2.2. Using the SELECT group, create the same three variables in *Exercise 2.1.*

Exercise 2.3. Instead of using the multiple IF-THEN/ELSE statement to create variables MATH_POINT and ENGLISH_POINT in *Exercise 2.1*, use one IF-THEN/ELSE statement with the assignment statement to create this variable following the examples in Section 2.3.