Part 1

The Basics of Debugging

Chapter 1

Understanding the Types of Errors in SAS Programs

Introduction

A variety of errors can occur in a SAS program, and these errors can occur at any stage of processing a program. Understanding the types of errors found in SAS programs can help you efficiently correct your program.

SAS detects some errors for you. For example, SAS finds misspelled keywords and invalid options and writes messages to the SAS log describing the problems it discovers. SAS finds these errors during compilation and prevents the program from executing.

The errors that SAS cannot detect for you are the logic errors that you code into your programming statements. To correct these errors, you must understand your data and review the processing notes and output generated by your program. A program with logic errors may execute and the SAS log may show that no errors occurred during the processing. It is up to you to review the results to determine if your program executed correctly.

This chapter describes the categories of errors that occur in SAS programs. Understanding the content of this chapter requires that you understand the basics of SAS processing. To review these concepts, refer to Appendix 1, “Review of SAS Processing Concepts.”

What Errors Does SAS Automatically Find for Me?

SAS looks for four types of errors in your programs. These four types are

- Syntax errors: occur when a program statement does not conform to the rules of the SAS language (e.g., a variable name starts with a number)
- Semantic errors: occur when a program statement is syntactically correct, but the structure of the statement is incorrect (e.g., an array reference was not specified correctly)
- Execution-time errors: occur when compiled statements are applied to data values (e.g., division by zero)
- Data errors: occur when the statements are correct, but the data is invalid (e.g., taking the logarithm of zero).

Syntax Errors

Syntax errors occur when program statements do not conform to the rules of the SAS language. Most of these errors are up to you to correct. Occasionally SAS makes assumptions about the context of the error, corrects the statement, and continues processing the program.

Common syntax errors include misspellings of keywords or omission of semicolons.

When SAS detects a syntax error, it first tries to correct the error by deleting, inserting, or replacing tokens in the SAS statement. If this action succeeds, SAS notifies you about the action it took by underlining the code where it detected the syntax error and by writing a message to the SAS log about the error and the action taken. SAS continues processing the step.

When SAS cannot correct a syntax error it finds, it stops processing the step. Processing resumes with the next step in the program. The results of the subsequent steps, however, may be incorrect if information in these steps was needed from the step containing the error.

Six examples of syntax errors follow.

Example 1.1: Identifying a Syntax Error in a DATA Step That SAS Can Correct

This example shows that SAS can correctly determine the correct syntax for a specific syntax error it detected in a DATA step.

This DATA step reads specific observations from CORPLIB.ITEMS and creates a new variable, PROJCOST.

The error in the program is the misspelling of the keyword ATTRIB.

Original Program

```
data temp;
  set corplib.items(where=(itemtype='S'));

  attrb projcost format=dollar10.2 label='Projected Cost';

  projcost=110*itemcost;
run;
```

The SAS log for this program follows. It shows that SAS correctly fixed the error in the ATTRIB statement and that it executed the DATA step.

SAS Log

```
data temp;
11     set corplib.items(where=(itemtype='S'));
12
13     attrb projcost format=dollar10.2 label='Projected Cost';
       -----
       1
WARNING 1-322: Assuming the symbol ATTRIB was misspelled as attrb.

14
15     projcost=110*itemcost;
16   run;

NOTE: There were 751 observations read from the dataset
  CORPLIB.ITEMS.
      WHERE itemtype='S';
NOTE: The data set WORK.TEMP has 751 observations and 15 variables.
NOTE: DATA statement used:
      real time 0.06 seconds
```

Example 1.2: Identifying a Syntax Error in a DATA Step That SAS Cannot Correct

This example demonstrates that SAS cannot always determine how to correct your syntax errors.

This DATA step reads specific observations from CORPLIB.ITEMS and creates a new variable, PROJCOST.

The syntax error in the program is that the semicolon that should end the SET statement is missing.

Original Program

```
data temp;
  set corplib.items(where=(itemtype='S'))

  attrib projcost format=dollar10.2 label='Projected Cost';

  projcost=110*itemcost;
run;
```

The SAS log for this program follows. Callouts identify important features in the log. A description of the features follows the log.

The first error message gives you a place to begin your search for the error in your program. It tells you where SAS expected to find a token different from what it found.

SAS Log

```
28  data temp;
29    set corplib.items(where=(itemtype='S'))
30
31    attrib projcost format=dollar10.2 label='Projected Cost';
                      ------
    ❶                 22
                      76
ERROR 22-322: Syntax error, expecting one of the following: a name,
  a quoted string, (, ;, END, KEY, KEYS, NOBS, OPEN, POINT, _DATA_,
  _LAST_, _NULL_.

ERROR 76-322: Syntax error, statement will be ignored.

31
32   projcost=110*itemcost;
33   run;

NOTE: The SAS System stopped processing this step because of
      errors.
 ❷
WARNING: The data set WORK.TEMP may be incomplete.  When this step
  was stopped there were 0 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
```

❶ SAS tokenizes the DATA step. SAS underlines the first token out of place, FORMAT=. The text of the error message lists several keywords associated with DATA statements and SET statements. Therefore, the error is likely in either the DATA statement or the SET statement. It appears that SAS considers FORMAT= to be part of the SET statement and ATTRIB and PROJCOST to be data set names.

❷ The WARNING informs you that SAS created the data set WORK.TEMP, but that it may be incomplete because SAS stopped the step as a result of the error. The data set WORK.TEMP contains no observations.

Example 1.3: Identifying Incorrect Repair of a Syntax Error in a PROC SQL Step

This example demonstrates that SAS sometimes repairs syntax errors incorrectly.

This PROC SQL step should select specific observations and variables from CORPLIB.ITEMS and present the selections ordered by the variable AUTHOR.

The syntax error in the PROC SQL step is the misspelled keyword ORDER. The misspelling is such that SAS interprets the token as the operator OR rather than the keyword ORDER.

Original Program

```
proc sql;
  select author,title,pubyear
  from corplib.items
  where pubyear=1999 and libsect='General'
  orer by author;
quit;
```

The SAS log shows where SAS misinterprets the misspelling. This misinterpretation causes more errors, because SAS now interprets that the BY keyword is a variable and part of the WHERE clause. It looks for an operator following BY to complete the WHERE condition.

SAS Log

```
431  proc sql;
432    select author,title,pubyear
433    from corplib.items
434    where pubyear=1999 and libsect='General'
435    orer by author;
       ----    ------
       1       22
               76
WARNING 1-322: Assuming the symbol OR was misspelled as orer.

ERROR 22-322: Syntax error, expecting one of the following: !, !!,
  &, *, **, +, -, '.', /, <, <=, <>, =, >, >=, ?, AND, CONTAINS, EQ,
  GE, GROUP, GT, HAVING, LE, LIKE, LT, NE, OR, ORDER, ^=, |, ||, ~=.

ERROR 76-322: Syntax error, statement will be ignored.

436  quit;
NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: PROCEDURE SQL used:
      real time           0.00 seconds
```

Example 1.4: Identifying a Syntax Error in a Procedure That SAS Cannot Correct

This example demonstrates that SAS cannot always determine how to correct your syntax errors.

SAS lists syntax errors found in PROC steps the same way as it does for DATA steps. SAS identifies the error and writes a message to the SAS log. Syntax errors in procedures include misspelled keywords and missing semicolons. An example of a syntax error follows.

The report produced by this PROC TABULATE step should not include the horizontal separator lines in the row titles and the body of the table. The option NOSEPS suppresses these separators. This option is misspelled as NO SEPS in the example.

Original Program

```
proc tabulate data=corplib.items no seps;
  class libsect
  var itemcost;
  tables libsect all='Total for All Sections',
         itemcost*(n*f=5. (mean sum)*f=dollar8.2);
run;
```

The SAS log for this program follows. SAS identifies the error and stops the step.

SAS Log

```
44   proc tabulate data=corplib.items no seps;
                                      --
                                      22
                                         ----
                                         202
ERROR 22-322: Syntax error, expecting one of the following: ;,
              (, CLASSDATA, CONTENTS, DATA, DEPTH, EXCLNPWGT,
              EXCLNPWGTS, EXCLUSIVE, FORMAT, FORMCHAR,
              MISSING, NOSEPS, ORDER, OUT, PCTLDEF, QMARKERS,
              QMETHOD, QNTLDEF, STYLE, TRAP, VARDEF.
ERROR 202-322: The option or parameter is not recognized and
               will be ignored.
45     class libsect;
46     var itemcost;
47     tables libsect all='Total for All Sections',
48            itemcost*(n*f=5. (mean sum)*f=dollar8.2);
49   run;
```

SAS Log (*continued*)

```
NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: PROCEDURE TABULATE used:
      real time           0.04 seconds
```

Example 1.5: Identifying an Invalid Request of a SAS Procedure

This example presents a syntax error that prevents the PROC step from executing. The error is the omission of required specifications for an option.

The PROC FREQ step specifies that the frequencies for EMPSTATE not be printed. The step does not include code to create an output data set. With the NOPRINT option in effect and no output data set requested, SAS does not process the step.

Original Program

```
proc freq data=corplib.employees;
  tables empstate / noprint;
run;
```

The SAS log shows the warning SAS produces when this program executes. The SAS language statements follow the rules, but the request is invalid.

SAS Log

```
53   proc freq data=corplib.employees;
54     tables empstate / noprint;
55   run;

WARNING: There are no valid requests for output data sets or
         printed output, so processing will terminate.
NOTE: PROCEDURE FREQ used:
      real time           0.04 seconds
```

The program can be corrected either by removing the NOPRINT option or by adding the OUT= option. The PROC FREQ step below includes the OUT= option.

Revised Program

```
proc freq data=corplib.employees;
  tables empstate / noprint out=statefreqs;
run;
```

Example 1.6: Identifying a Syntax Error in Macro Language

This example shows how the macro processor detects a syntax error in macro language.

This macro program should produce a PROC TABULATE step that summarizes data in CORPLIB.CIRCUL for a specific year. The parameter to the program is the year of the analysis.

The problem with the macro program is the incorrect specification of the macro program name in the %MACRO statement. The error in the name is the space. The macro processor cannot tokenize the macro program because of this syntax error.

Original Program

```
%macro annual_circulation_ statistics(year);
  proc tabulate data=corplib.circul;
    where year(checkout)=&year;
    title "Circulation Statistics for &year";
    class nrenew;
    tables nrenew all,n*f=6.;
  run;
%mend annual_circulation_statistics;
```

The SAS log for compilation of the macro program follows. The first error message indicates that a semicolon is missing from a macro language statement. The note at the end of the macro program provides a clue that something is wrong with the %MACRO statement, because it states that the macro program name is ANNUAL_CIRCULATION_.

SAS Log

```
ERROR: Expected semicolon not found.  The macro will not be
       compiled.
ERROR: A dummy macro will be compiled.
205  %macro annual_circulation_ statistics(year);
206    proc tabulate data=corplib.circul;
207      where year(checkout)=&year;
208      title "Circulation Statistics for &year";
209      class nrenew;
210      tables nrenew all,n*f=6.;
211    run;
212  %mend annual_circulation_statistics;
NOTE: Extraneous information on %MEND statement ignored for
      macro definition ANNUAL_CIRCULATION_.
```

This example generates a second error message as well. The "dummy macro" is a macro program that the macro processor compiles but does not store and does not execute. The macro processor creates this dummy macro program when it finds a syntax error.

Semantic Errors

A semantic error occurs when the structure of a SAS statement is correct, but elements in the statement are not valid for that usage. A semantic error is like a sentence in which the subject and the verb don't agree:

```
The sky are cloudy.
```

The compiler detects semantic errors. SAS does not detect semantic errors during tokenization because nothing is wrong with the tokens. The problem is that SAS does not know how to interpret your code.

Typical semantic errors include misspellings of variable names and incorrect specifications of arrays. Three examples follow.

Example 1.7: Identifying a DATA Step with a Semantic Error

This example presents a DATA step containing a semantic error that stops the DATA step from completely executing. The code is syntactically correct and compiles without errors, but execution of the code is not possible.

This DATA step reads usage information for items for the four years 1999 through 2002. An item's four years of information is on one data line. The goal is to create four observations for each item, one for each year.

For example, the first data line should produce these four observations:

Obs	ITEMID	ITEM YEAR	NCHECKOUT
1	LIB0784	1999	3
2	LIB0784	2000	10
3	LIB0784	2001	0
4	LIB0784	2002	2

The semantic error is the incomplete specification of the ITEMOUT array in the DO loop.

Original Program

```
data itemusage;
  input itemid $ 1-7 n1999 n2000 n2001 n2002;

  array itemout{4} n1999-n2002;
  keep itemyear ncheckout;

  do i=1 to 4;
    itemyear=i+1998;
    ncheckout=itemout;
    output;
  end;
datalines;
LIB0784 3 10 0 2
LIB0785 0 1 0 0
LIB0786 11 14 23 18
run;
```

The SAS log shows that the DATA step could be tokenized and that the step did not contain any syntax errors. An ERROR message states that the reference to the array on line 277 cannot be resolved. This type of message indicates a semantic error.

SAS Log

```
269  data itemusage;
270    input itemid $ 1-7 n1999 n2000 n2001 n2002;
271
272    array itemout{4} n1999-n2002;
273    keep itemyear ncheckout;
274
275    do i=1 to 4;
276      itemyear=i+1998;
277      ncheckout=itemout;
ERROR: Illegal reference to the array itemout.
278      output;
279    end;
280  datalines;

NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.ITEMUSAGE may be incomplete.  When
this step was stopped there were 0 observations and 2 variables.
NOTE: DATA statement used:
      real time           0.11 seconds
run;
```

The corrected statement in the DO loop referencing ITEMOUT follows.

```
ncheckout=itemout{i};
```

Example 1.8: Identifying a DATA Step with a Semantic Error

This example presents a DATA step containing a semantic error that stops the DATA step from completely executing. The code is syntactically correct and compiles without errors, but execution of the code is not possible.

This DATA step creates an output data set containing observations from CORPLIB.ITEMS where the value of the variable ITEMCOST is greater than 100.

The semantic error in this DATA step is that the data set name in the DATA statement and the OUTPUT statement do not agree.

The OUTPUT statement directs that observations be written to a specific data set. That data set name must be explicitly named in the DATA statement.

Original Program

```
data tmp;
  set corplib.items;

  if itemcost > 100 then output temp;
run;
```

The SAS log identifies the mismatch between the data set name in the DATA statement and the data set name following OUTPUT. SAS cannot correct this kind of error; it can only tell you where it thinks the problem is.

SAS Log

```
285  data tmp;
286    set corplib.items;
287
288    if itemcost > 100 then output temp;
                                    ----
                                    455
ERROR 455-185: Dataset was not specified on the DATA statement.

289  run;

NOTE: The SAS System stopped processing this step because of
      errors.
WARNING: The data set WORK.TMP may be incomplete.  When this step
  was stopped there were 0 observations and 13 variables.
NOTE: DATA statement used:
      real time           0.04 seconds
```

Example 1.9: Incorrectly Specifying a PROC REPORT Statement

This example presents a PROC step with a semantic error.

The goal of this PROC step is to list the number of orders per distributor and the total cost of all orders per distributor. The syntax of the PROC REPORT step is correct, but the DEFINE statement for DISTID is incorrectly specified. The variable DISTID is character and cannot be specified as an analysis variable.

Original Program

```
proc report data=corplib.orders box nowindows;
  column distid ordertot;
  define distid    / analysis n 'Distributor' width=11;
  define ordertot  / analysis sum 'Total Cost' width=11
                     format=dollar10.2;
run;
```

The SAS log for this PROC step follows. The error message identifies the problem in the DEFINE statement with the variable DISTID.

SAS Log

```
86   proc report data=corplib.orders box nowindows;
87     column distid ordertot;
88     define distid    / analysis n 'Distributor' width=11;
89     define ordertot  / analysis sum 'Total Cost' width=11
90                        format=dollar10.2;
91   run;

ERROR: distid is an ANALYSIS variable but not numeric.
NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: PROCEDURE REPORT used:
      real time           0.00 seconds
```

One way to correct this program follows. The variable DISTID is now defined as a GROUP variable. A new column, N, shows the total number of orders per DISTID.

Revised Program

```
proc report data=corplib.orders box windows;
  column distid n ordertot;
  define distid    / group 'Distributor' width=11;
  define n / "Number of Orders" width=6;
  define ordertot  / analysis sum 'Total Cost' width=11
                     format=dollar10.2;
run;
```

Execution-Time Errors

SAS detects execution-time errors when it applies compiled programming statements to data values. Typical execution-time errors include

- INPUT statements that do not match the data lines
- illegal mathematical operations such as division by zero
- observations not sorted in the order specified in the BY statement when doing BY-group processing
- reference to a nonexistent member of an array
- illegal arguments to functions
- no resources to complete a task specified in the program.

When SAS encounters execution-time errors, it usually produces warning messages and continues to process the program. The information that SAS writes to the SAS log includes the following:

- an error message
- the values stored in the input buffer if SAS is reading data values from a source other than a data set (e.g., an external file)
- the contents of the program data vector at the time the error occurred
- a message explaining the error.

Errors that occur prior to the PROC step may be the source of an error in a procedure step. If the procedure depends on data generated in previous steps, the procedure may not execute and SAS may generate an error. Depending on the procedure, incorrect data may not stop the processing of the procedure, but the results instead will be in error.

Note that the information written to the SAS log varies depending on how you specify certain SAS options. These options are described in Chapter 2.

Since execution-time errors result from applying your compiled SAS statements to your data values, you must understand the data you are processing in order to correct the errors.

Three examples of execution-time errors follow.

Example 1.10: Identifying an Execution-Time Error in a DATA Step Caused by Division by Zero

This example shows how SAS processes a DATA step that contains an execution-time error. In this example, but not in all DATA steps with execution-time errors, the DATA step compiles correctly and executes completely.

This DATA step reads in three data lines and does a computation. The value for NRECVD in the second data line is zero. Since NRECVD is the divisor in the computation, this statement does not execute for the second observation. Division by zero in the second data line is the execution-time error in this DATA step.

Original Program

```
data newbooks;
  input title $ 1-20 totlcost nrecvd;
  attrib nrecvd   label='Number of Copies Received'
         totlcost label='Total Cost for Title'
         cost     label='Cost Per Copy Received';
  cost=totlcost/nrecvd;
datalines;
My Computer          28.63 3
Business 101         56.33 0
Ergonomic Offices    73.98 1
run;
proc print data=newbooks;
  title 'New Books Ordered';
run;
```

The SAS log for this program identifies the record where the division by zero occurred. Processing does not stop when SAS encounters this type of error. For the data line in error, SAS assigns a missing value to COST. SAS describes this action in the SAS log.

SAS Log

```
66   data newbooks;
67     input title $ 1-20 totlcost nrecvd;
68     attrib nrecvd   label='Number of Copies Received'
69            totlcost label='Total Cost for Title'
70            cost     label='Cost Per Copy Received';
71     cost=totlcost/nrecvd;
72   datalines;

NOTE: Division by zero detected at line 71 column 16.
RULE:----+----1----+----2----+----3----+----4----+----5----+
74   Business 101         56.33 0
title=Business 101 totlcost=56.33 nrecvd=0 cost=. _ERROR_=1 _N_=2
NOTE: Mathematical operations could not be performed at the
      following places. The results of the operations have
      been set to missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 71:16
```

SAS Log (*continued*)

```
NOTE: The data set WORK.NEWBOOKS has 3 observations and 4
      variables.
NOTE: DATA statement used:
      real time           0.15 seconds
76   run;
77   proc print data=newbooks;
78     title 'New Books Ordered';
79   run;

NOTE: There were 3 observations read from the dataset
      WORK.NEWBOOKS.
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
```

The following PROC PRINT output verifies that the value of COST is missing in the second data line.

Output

```
                         New Books Ordered                  6

    Obs    title                totltcost    nrecvd      cost

      1    My Computer            28.63         3       9.5433
      2    Business 101           56.33         0        .
      3    Ergonomic Offices      73.98         1      73.9800
```

One way to avoid the error is to test the value of NRECVD and to compute COST only when NRECVD is greater than zero. The following IF statement accomplishes this and prevents the execution-time error.

```
if nrecvd > 0 then cost=totlcost/nrecvd;
```

Example 1.11: Identifying an Execution-Time Error Caused by Errors in BY-Group Specifications

This example shows how SAS processes a program that contains an execution-time error. The DATA step where SAS detects the error compiles correctly but does not completely execute.

This program sorts a data set and summarizes the data set in a DATA step according to BY-group values. The goal of the program is to determine for each author the number of titles with multiple copies.

The list of variables in the BY statement in the PROC SORT step and the list of variables in the BY statement in the DATA step do not agree. This is an example of an execution-time error that stops the processing of a step.

Original Program

```
proc sort data=corplib.items;
  by author;
run;
data temp;
  set corplib.items;
  by author title;

  attrib nmult length=4
         label='Number of Titles with Multiple Copies';

  if first.title then do;
    nmult=0;
    if not last.title then nmult+1;
  end;

  if last.author then output;
run;
```

SAS writes a message to the SAS log describing the error. SAS displays the values of the variables for the observation where it first encounters the problem. Following these are the notes and warnings that describe the action that SAS took with regard to this problem. Here, SAS stopped the DATA step and the resulting data set, WORK.TEMP, is incomplete.

SAS Log

```
290  proc sort data=corplib.items;
291    by author;
292  run;

NOTE: There were 3750 observations read from the dataset
      CORPLIB.ITEMS.
NOTE: The data set CORPLIB.ITEMS has 3750 observations and 14
      variables.
NOTE: PROCEDURE SORT used:
      real time           0.04 seconds

293  data temp;
294    set corplib.items;
295    by author title;
296
```

SAS Log (*continued*)

```
  attrib nmult length=4
         label='Number of Titles with Multiple Copies';

  if first.title then do;
    nmult=0;
    if not last.title then nmult+1;
  end;

  if last.author then output;
run;

ERROR: BY variables are not properly sorted on data set
  CORPLIB.ITEMS.
itemid=LIB0859 title=Title 2-1E3 author=Last1E3,  First1E3 copynum=2
  callnum=110.28 publish=Publisher 9 pubcity=City 9 pubyear=2004
  libsect=Serials itemtype=S orderdat=. ordernum=.
itemcost=. subscdat=. FIRST.author=0 LAST.author=0 FIRST.title=1
  LAST.title=0 nmult=1 _ERROR_=1 _N_=162
NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: There were 163 observations read from the dataset
  CORPLIB.ITEMS.
WARNING: The data set WORK.TEMP may be incomplete.  When this step
  was stopped there were 111 observations and 15 variables.
NOTE: DATA statement used:
      real time           0.04 seconds
```

Correct the program by modifying the BY statement in the PROC SORT step as follows. The DATA step remains the same.

Revised Program

```
proc sort data=corplib.items;
  by author title;
run;
```

Example 1.12: Identifying a PROC Step That Executes but Does Not Produce Expected Results

This example shows that a PROC step with execution-time errors may execute completely, but it will not necessarily produce the expected results. This demonstrates the importance of careful review of the SAS log to determine that an error occurred.

The semicolon is missing in the TITLE statement. Because of the TITLE statement's position, the CLASS statement becomes part of the TITLE statement. Statistics then are computed overall rather than by the categories of LIBSECT.

Original Program

```
proc means data=corplib.items;
  title "Means by Library Section"
  class libsect;
  var itemcost;
run;
```

A warning message does indicate that there may be a problem with the TITLE statement.

SAS Log

```
64   proc means data=corplib.items;
65     title "Means by Library Section"
66     class libsect;
WARNING: The TITLE statement is ambiguous due to invalid
         options or unquoted text.
67     var itemcost;
68   run;

NOTE: There were 3750 observations read from the dataset
      CORPLIB.ITEMS.
NOTE: PROCEDURE MEANS used:
      real time           0.04 seconds
```

Data Errors

A data error occurs when a data value is not appropriate for the SAS statements you coded. SAS detects these errors when the program executes, but these errors are different from execution-time errors. With execution-time errors, something in the program statements is wrong. With data errors, the data is wrong. Data errors reflect problems with the creation of the input data source.

Remedies for data errors include correcting the data entry process or changing the DATA step to reflect the chance that errors may occur.

When SAS detects a data error, it writes a message to the SAS log, lists the values of the input buffer when it reads raw data, and lists the program data vector for the observation where the error occurred.

One example follows.

Example 1.13: Identifying Input Data That Generates a Data Error

This example presents a DATA step where a data value is in error and the DATA step code is not. It demonstrates that you may need to include statements to handle potential errors to ensure that the DATA step executes as requested.

This DATA step reads in four data lines. The second variable is a date, and the date of January 32, 2001, on the third data line is invalid. This invalid date generates a data error.

Original Program

```
data temp;
  input empno 6. +1 checkout mmddyy10.;
datalines;
002020 01152001
002043 01162001
002087 01322001
002218 01232001
run;
proc print data=temp;
  format checkout mmddyy10.;
run;
```

The SAS log for this program identifies the data line containing the problem.

SAS Log

```
445  data temp;
446    input empno 6. +1 checkout mmddyy10.;
447  datalines;

NOTE: Invalid data for checkout in line 450 8-17.
RULE:----+----1----+----2----+----3----+----4----+----5----+--
450  002087 01322001
empno=2087 checkout=. _ERROR_=1 _N_=3
NOTE: The data set WORK.TEMP has 4 observations and 2 variables.
```

SAS Log (*continued*)

```
NOTE: DATA statement used:
      real time           0.06 seconds

452  run;
453  proc print data=temp;
454    format checkout mmddyy10.;
455  run;

NOTE: There were 4 observations read from the dataset
      WORK.TEMP.
NOTE: PROCEDURE PRINT used:
      real time           0.05 seconds
```

The following PROC PRINT output shows that SAS read the four data lines and set the value of CHECKOUT to missing for the third observation.

Output

```
                          The SAS System                      1

                   Obs     empno        checkout

                    1       2020      01/15/2001
                    2       2043      01/16/2001
                    3       2087               .
                    4       2218      01/23/2001
```

One way to correct this program is to read the month, day, and year separately and test for valid values. When a value is valid, assign it to CHECKOUT using the MDY function. If invalid, the value of CHECKOUT remains missing. A corrected program follows.

Revised Program

```
data temp;
  input empno 6. +1 cmonth 2. cday 2. cyear 4.;

  if (1 le cmonth le 12) and
     ((1 le cday le 28) or
     ((29 le cday le 30) and cmonth ne 2) or
     (cday=31 and cmonth in (1,3,5,7,8,10,12)) or
     (cday=29 and cmonth=2 and mod(cyear,4)=0)) then
     checkout=mdy(cmonth,cday,cyear);
datalines;
002020 01152001
002043 01162001
002087 01322001
002218 01232001
run;
```

What Errors Doesn’t SAS Find for Me?

According to SAS, your program is syntactically correct and executes without error. Your SAS log does not contain messages describing syntax errors, semantic errors, execution-time errors, or data errors. Yet, when you review the results of your SAS program, you find them incorrect. This happens when your instructions to SAS did not correctly convey the actions you wanted SAS to take. Your program executes anyway because SAS does not know how to detect when your logic is faulty.

TIP! How can you find the logic errors that you introduce into your programs? The answer is by fully understanding your data, closely reading the messages in the SAS log, carefully reviewing the results, and using the tools that SAS provides for detecting logic errors.

Chapters 3 and 4 describe the tools for finding logic errors in SAS language. Chapter 5 describes the tools in the macro facility.

Example 1.14: Identifying a DATA Step with a Logic Error

The program in this example contains a logic error in the DATA step. The program compiles correctly and executes without errors. The example demonstrates that careful review of the notes and the output may be necessary to identify incorrect results of your program.

This program merges the distributor information with the order information and retains information only for distributor ‘D001’. The data sets should be matched by the distributor ID, DISTID.

A logic error exists in the DATA step. The program executes without errors, as shown in the SAS log, but the output shows that the merge did not execute as required. Distributor information for distributor ‘D001’ is not merged with the order information as required.

The problem with the DATA step is that a BY statement directing a matched merge on DISTID is missing. Without a BY statement, SAS does a one-to-one merge. SAS, however, does not know that this is an error.

Original Program

```
proc sort data=corplib.distrib;
  by distid;
run;
proc sort data=corplib.orders;
  by distid
run;
data distid1;
```

Original Program (*continued*)

```
  merge corplib.distrib corplib.orders;
  if distid='D001';
run;
proc print data=distid1;
  title "Orders for Distributor D001";
run;
```

The SAS log for the DATA step shows that it executed without error.

SAS Log

```
57   data distid1;
58     merge corplib.distrib corplib.orders;
59
60     if distid='D001';
61   run;

NOTE: There were 5 observations read from the data set
      CORPLIB.DISTRIB.
NOTE: There were 53 observations read from the data set
      CORPLIB.ORDERS.
NOTE: The data set WORK.DISTID1 has 7 observations and 13
      variables.
NOTE: DATA statement used:
      real time           0.10 seconds
```

Reviewing the results shows that the DATA step included information about distributors other than D001 in WORK.DISTID1. The data set CORPLIB.DISTRIB contains information about five distributors, and the data set CORPLIB.ORDERS contains information about seven orders for D001. The resulting data set WORK.DISTID1 does contain seven observations, but all except the first are in error. Observations 2 through 5 contain address information for distributors other than D001. Observations 6 and 7 contain no distributor addresses.

Output

```
                  Orders for Distributor D001

Obs   distid      distname         distaddr        distcity

 1     D001     Distributor 1    Distributor 1   Distributor 1
 2     D001     Distributor 2    Distributor 2   Distributor 2
 3     D001     Distributor 3    Distributor 3   Distributor 3
 4     D001     Distributor 4    Distributor 4   Distributor 4
 5     D001     Distributor 5    Distributor 5   Distributor 5
 6     D001
 7     D001
```

Output (*continued*)

```
Obs   diststat   distzip    distphon      distfax    ordernum

 1       NY       13021    3155555555   3155555555     0003
 2       IL       60000    3125555555   3125555555     0009
 3       CA       94000    6515555555   6515555555     0021
 4       MN       55100    6515555555   6515555555     0024
 5       PA       19000    9995555555   9995555555     0032
 6                    .             .            .     0039
 7                    .             .            .     0042

Obs     ordrdate    nitems      ordertot      datercvd

 1    02/28/1999       14        $458.17    04/25/1999
 2    07/21/1999        9        $274.66    08/16/1999
 3    01/03/2000       14        $512.78    01/16/2000
 4    02/03/2000        4        $111.92    03/11/2000
 5    04/06/2000       15        $554.96    04/07/2000
 6    07/12/2000        9        $257.99    08/27/2000
 7    09/06/2000       11        $430.04    10/10/2000
```

Adding a BY statement to the DATA step corrects the error.

Revised Program

```
data distid1;
  merge corplib.distrib corplib.orders;
    by distid;
  if distid='D001';
run;
```

Example 1.15: A PROC STEP That Executes, but Not on the Data Set Expected

This example demonstrates the importance of reviewing the SAS log and the output to determine if the program executed as required. The program compiles correctly and executes without error, but the results are not as expected.

This program reads the circulation statistics data set and computes the number of days an item has been checked out. These data are then matched against the CORPLIB.ITEMS data set to find the library section for the item. The PROC MEANS step computes statistics on the new variable DAYSOUT for each value of LIBSECT and saves the results in the temporary data set, WORK.CIRCSUMM.

The intent of the PROC PRINT step is to list each of the items in the CIRC data set that has been in circulation for at least 30 days. Since the DATA= option is missing from the PROC PRINT statement, PROC PRINT lists the data from the most recently created data set. In this program, the most recently created data set is the one created by PROC MEANS and not the WORK.CIRC data set.

Original Program

```
data circ;
  set corplib.circul;

  if checkin=. then daysout=today()-checkout;
  else daysout=checkin-checkout;
run;
proc sql;
  create table circ
    as select daysout,c.itemid,c.copynum,libsect
    from circ c left join corplib.items i
    on c.itemid=i.itemid;
quit;
proc means data=circ;
  class libsect;
  var daysout;
  output out=circsumm n=n mean=daysout;
run;
proc print;
  title "Items Checked Out Longer than 30 Days";
  where daysout > 30;
run;
```

The SAS log shows no errors because the variable name DAYSOUT has been assigned to the mean value of DAYSOUT in the PROC MEANS step. The note from the PROC PRINT step shows that the PROC PRINT output was produced from WORK.CIRCSUMM. The intent was to list with PROC PRINT the observations in WORK.CIRC .

SAS Log

```
111  data circ;
112    set corplib.circul;
113
114    if checkin=. then daysout=today()-checkout;
115    else daysout=checkin-checkout;
116  run;

NOTE: There were 446 observations read from the dataset
      CORPLIB.CIRCUL.
NOTE: The data set WORK.CIRC has 446 observations and 10
      variables.
```

SAS Log (*continued*)

```
NOTE: DATA statement used:
      real time           0.10 seconds

117  proc sql;
118    create table circ
119      as select daysout,c.itemid,c.copynum,libsect
120      from circ c left join corplib.items i
121      on c.itemid=i.itemid;
NOTE: Table WORK.CIRC created, with 2306 rows and 4 columns.
122  quit;
NOTE: PROCEDURE SQL used:
      real time           0.04 seconds

123  proc means data=circ;
124    class libsect;
125    var daysout;
126    output out=circsumm n=n mean=daysout;
127  run;

NOTE: There were 2306 observations read from the dataset
      WORK.CIRC.
NOTE: The data set WORK.CIRCSUMM has 7 observations and 5
      variables.
NOTE: PROCEDURE MEANS used:
      real time           0.06 seconds

128  proc print;
129    title "Items Checked Out Longer than 30 Days";
130    where daysout > 30;
131  run;

NOTE: There were 7 observations read from the dataset
      WORK.CIRCSUMM.
      WHERE daysout>30;
NOTE: PROCEDURE PRINT used:
      real time           0.16 seconds
```

To correct this program, modify the PROC PRINT statement as follows:

```
proc print data=circ;
```