临床研究SAS高级编程
—— SAS Data Step

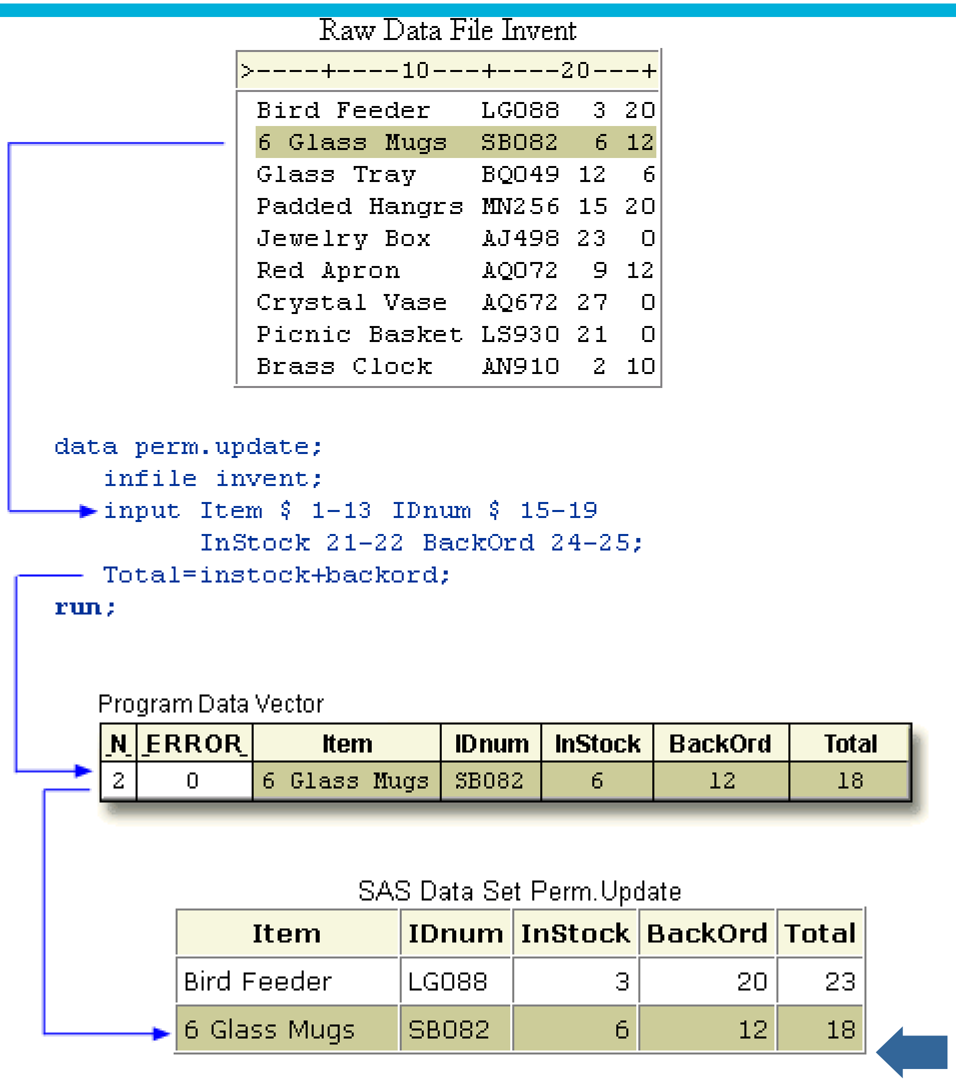
Contents

Overview

Introduction

- This section teaches you what happens "behind the scenes" when the DATA step reads raw data. You'll examine the **program data vector**, which is a logical framework that SAS uses when creating SAS data sets.

- Understanding how the program operates can help you to anticipate how variables will be created and processed, to plan your modifications, and to interpret and debug program errors. It also gives you useful strategies for preventing and correcting common DATA step errors.



Raw Data File Invent

```
>----+----10---+----20---+
 Bird Feeder   LG088  3 20
 6 Glass Mugs  SB082  6 12
 Glass Tray    BQ049 12  6
 Padded Hangrs MN256 15 20
 Jewelry Box   AJ498 23  0
 Red Apron     AQ072  9 12
 Crystal Vase  AQ672 27  0
 Picnic Basket LS930 21  0
 Brass Clock   AN910  2 10
```

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

Program Data Vector

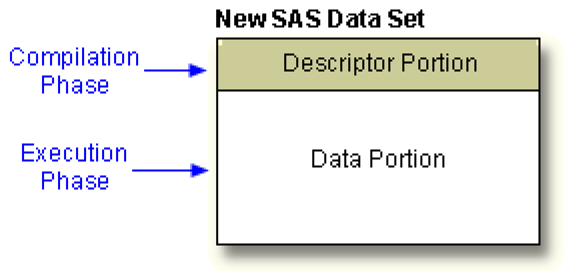
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	18

SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23
6 Glass Mugs	SB082	6	12	18

How SAS Processes Programs (1)

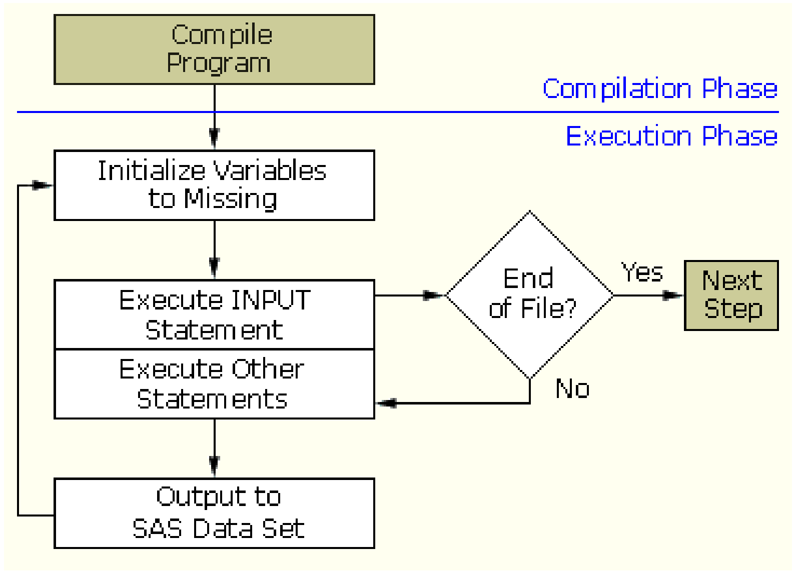
- When you submit a DATA step, SAS processes the DATA step and then creates a new SAS data set. Let's see exactly how that happens.
- A SAS DATA step is processed in two phases:



- During the **compilation phase**, each statement is scanned for syntax errors. Most syntax errors prevent further processing of the DATA step. When the compilation phase is complete, the descriptor portion of the new data set is created.
- If the DATA step compiles successfully, then the **execution phase** begins. During the execution phase, the DATA step reads and processes the input data. The DATA step executes once for each record in the input file, unless otherwise directed.

How SAS Processes Programs (2)

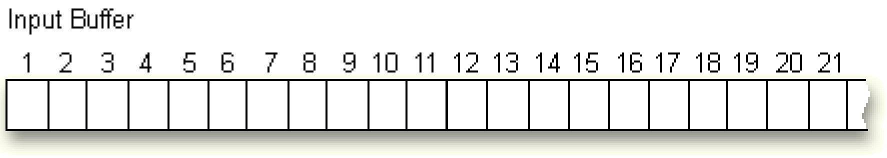
The diagram below shows the flow of DATA step processing for reading raw data. We'll examine both the compilation phase and the execution phase in this section.



Compilation Phase (1)

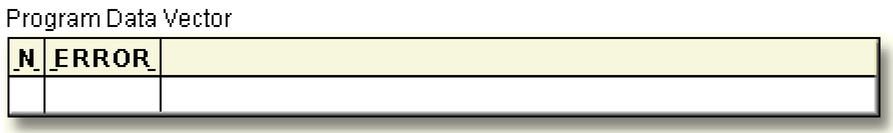
Input buffer

- At the beginning of the compilation phase, the **input buffer** (an area of memory) is created to hold a record from the external file. The input buffer is created only when raw data is read, not when a SAS data set is read. The term **input buffer** refers to a logical concept; it is not a physical storage area.



Compilation Phase (2)

- Program data vector
  - After the input buffer is created, the **program data vector** is created. The program data vector is the area of memory where SAS builds a data set, one observation at a time. Like the term **input buffer**, the term **program data vector** refers to a logical concept.
  - The program data vector contains two **automatic variables** that can be used for processing but which are not written to the data set as part of an observation.
    - _N_ counts the number of times that the DATA step begins to execute.
    - **_ERROR_** signals the occurrence of an error that is caused by the data during execution. The default value is *0*, which means there is no error. When one or more errors occur, the value is set to *1*.


Program Data Vector

_N_	_ERROR_	

Compilation Phase (3)

Syntax checking

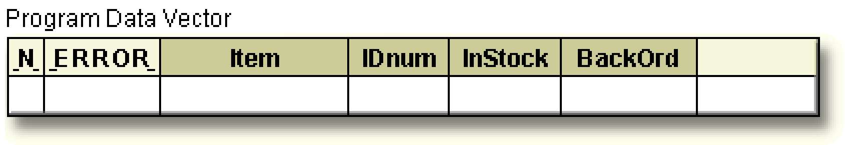
- During the compilation phase, SAS also scans each statement in the DATA step, looking for syntax errors. Syntax errors include
  - missing or misspelled keywords
  - invalid variable names
  - missing or invalid punctuation
  - invalid options.

Compilation Phase (4)

Data set variables (1)

- As the INPUT statement is compiled, a slot is added to the program data vector for each variable in the new data set. Generally, variable attributes such as length and type are determined the first time a variable is encountered.

```
data perm.update;
     infile invent;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```



Program Data Vector

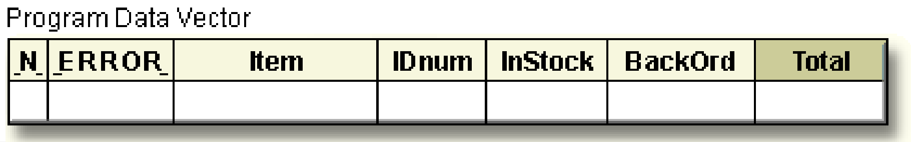
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	

Compilation Phase (5)

Data set variables (2)

- Any variables that are created with an assignment statement in the DATA step are also added to the program data vector. For example, the assignment statement below creates the variable Total. As the statement is compiled, the variable is added to the program data vector. The attributes of the variable are determined by the expression in the statement. Because the expression produces a numeric value, Total is defined as a numeric variable and is assigned the default length of 8.

```
data perm.update;
     infile invent;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```


Program Data Vector

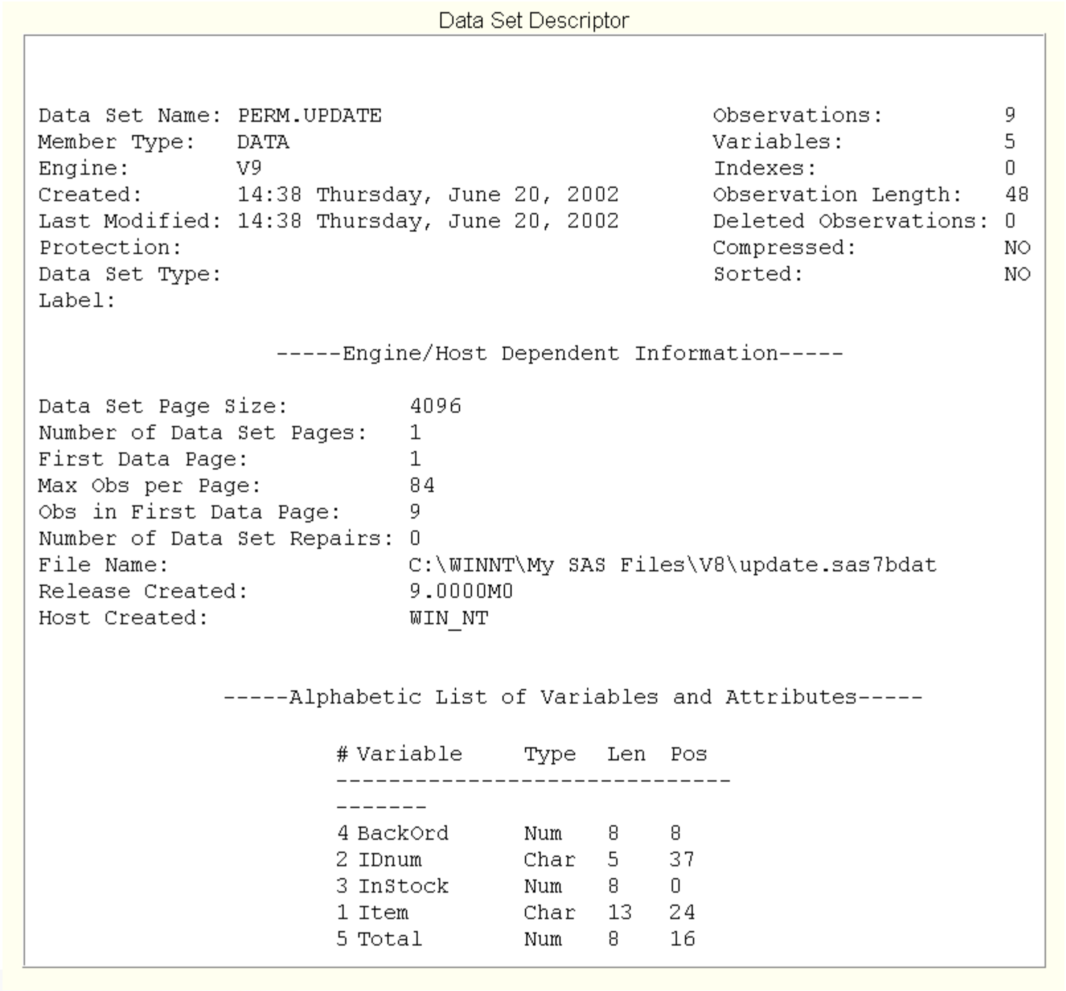
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total

Compilation Phase (6)

- Descriptor portion of the SAS data set (1)
  - At the bottom of the DATA step (in this example, when the RUN statement is encountered), the compilation phase is complete, and the descriptor portion of the new SAS data set is created. The descriptor portion of the data set includes
    - the name of the data set
    - the number of observations and variables
    - the names and attributes of the variables.
  - At this point, the data set contains the five variables that are defined in the input data set and in the assignment statement. Remember, _N_ and _ERROR_ are not written to the data set. There are no observations because the DATA step has not yet executed. During execution, each raw data record is processed and is then written to the data set as an observation.

Compilation Phase (7)

Descriptor portion of the SAS data set (2)



Data Set Descriptor

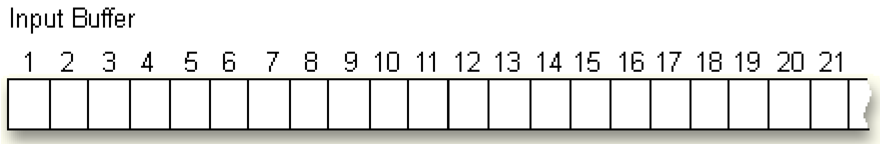
```
Data Set Name: PERM.UPDATE                                 Observations:          9
Member Type:   DATA                                        Variables:             5
Engine:        V9                                          Indexes:               0
Created:       14:38 Thursday, June 20, 2002               Observation Length:    48
Last Modified: 14:38 Thursday, June 20, 2002               Deleted Observations:  0
Protection:                                                Compressed:            NO
Data Set Type:                                             Sorted:                NO
Label:

                  -----Engine/Host Dependent Information-----

Data Set Page Size:          4096
Number of Data Set Pages:    1
First Data Page:             1
Max Obs per Page:            84
Obs in First Data Page:      9
Number of Data Set Repairs:  0
File Name:                   C:\WINNT\My SAS Files\V8\update.sas7bdat
Release Created:             9.0000M0
Host Created:                WIN_NT


              -----Alphabetic List of Variables and Attributes-----

                      # Variable     Type   Len   Pos
                      ------------------------------
                      -------
                      4 BackOrd      Num    8     8
                      2 IDnum        Char   5     37
                      3 InStock      Num    8     0
                      1 Item         Char   13    24
                      5 Total        Num    8     16
```

Compilation Phase (8)

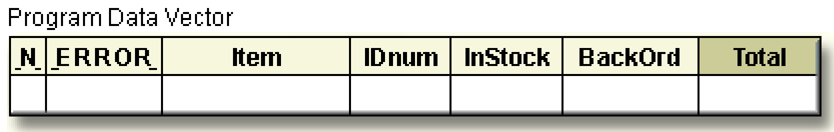
- Summary of the compilation phase
  - Let's review the compilation phase.

```
data perm.update;
     infile invent;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

  - During the compilation phase, the input buffer is created to hold a record from the external file.

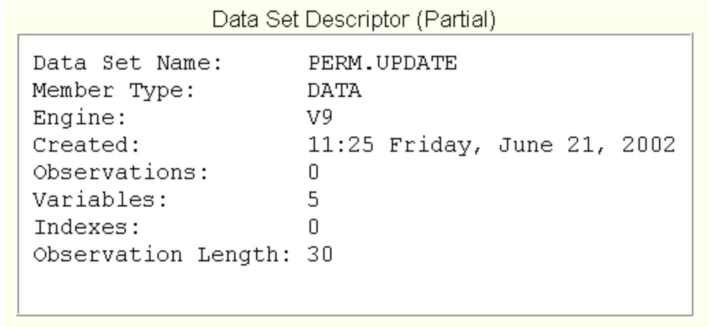


  - The program data vector is created to hold the current observation.


Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total

  - The descriptor portion of the SAS data set is created.


Data Set Descriptor (Partial)

```
Data Set Name:        PERM.UPDATE
Member Type:          DATA
Engine:               V9
Created:              11:25 Friday, June 21, 2002
Observations:         0
Variables:            5
Indexes:              0
Observation Length:   30
```

Quiz

- Which of the following is not created during the compilation phase?
  - a. the data set descriptor
  - b. the first observation
  - c. the program data vector
  - d. the _N_ and _ERROR_ automatic variables

  - Correct answer: b
    - At the beginning of the compilation phase, the program data vector is created. The program data vector includes the two automatic variables _N_ and _ERROR_. The descriptor portion of the new SAS data set is created at the end of the compilation phase. The descriptor portion includes the name of the data set, the number of observations and variables, and the names and attributes of the variables. Observations are not written until the execution phase.

Quiz

During the compilation phase, SAS scans each statement in the DATA step, looking for syntax errors. Which of the following is not considered a syntax error?

- a. incorrect values and formats
- b. invalid options or variable names
- c. missing or invalid punctuation
- d. missing or misspelled keywords

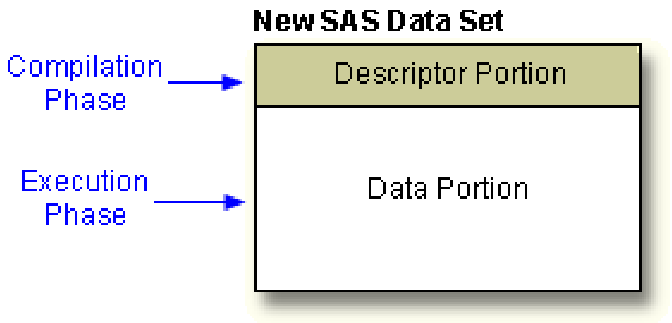
- Correct answer: a
  - Syntax checking can detect many common errors, but it cannot verify the values of variables or the correctness of formats.

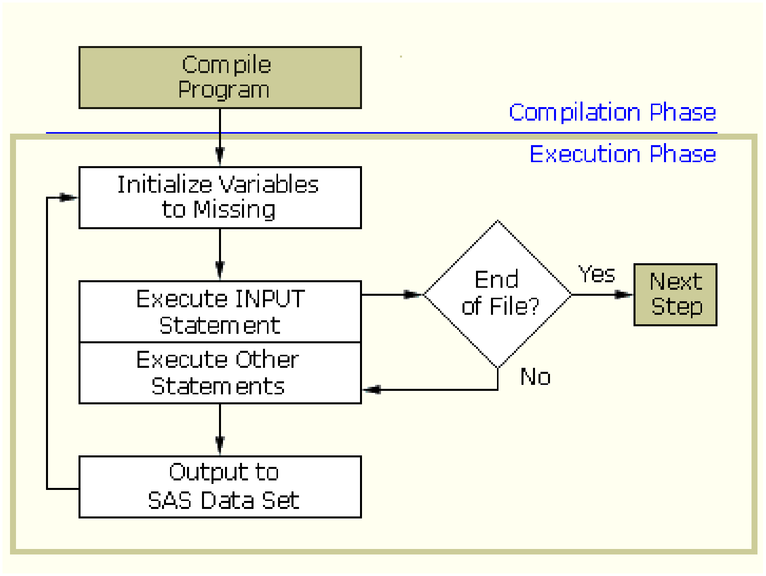
Execution Phase (1)

Overview (1)

- After the DATA step is compiled, it is ready for execution. During the execution phase, the data portion of the data set is created. The data portion contains the data values.

- During execution, each record in the input raw data file is read, stored in the program data vector, and then written to the new data set as an observation. The DATA step executes once for each record in the input file, unless otherwise directed by additional statements.



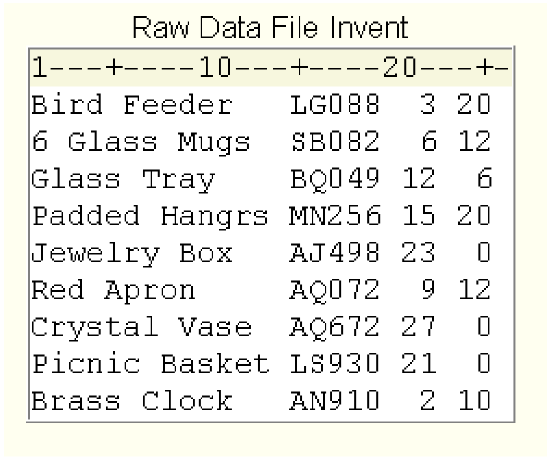


Execution Phase (2)

Overview (2)

- Example: The following DATA step reads values from the file **Invent** and executes nine times because there are nine records in the file.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
            InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```



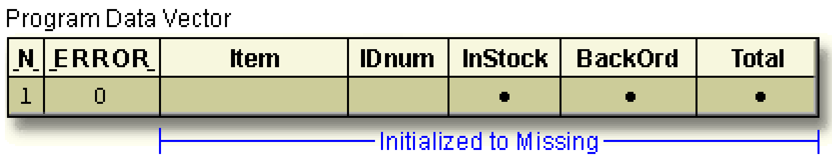
Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```

Execution Phase (3)

- Initializing variables
  - At the beginning of the execution phase, the value of _N_ is *1*. Because there are no data errors, the value of _ERROR_ is *0*.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```


Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0			•	•	•

Initialized to Missing

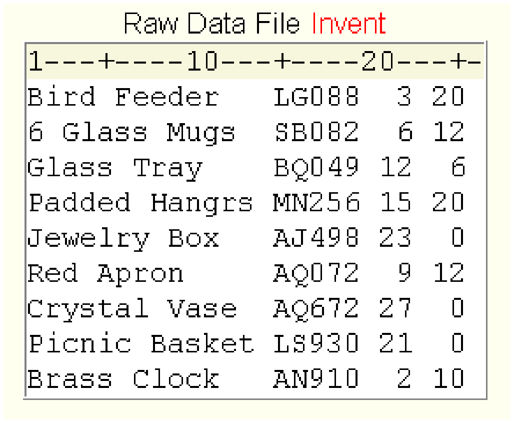
The remaining variables are initialized to missing. Missing numeric values are represented by periods, and missing character values are represented by blanks.

Execution Phase (4)

Input data

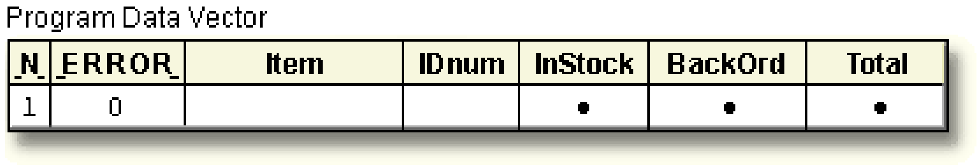
- Next, the INFILE statement identifies the location of the raw data.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```



Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```



Program Data Vector

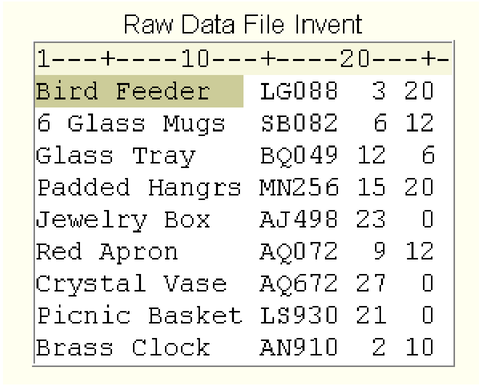
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0			•	•	•

Execution Phase (5)

Input pointer (1)

- When an INPUT statement begins to read data values from a record that is held in the input buffer, it uses an input pointer to keep track of its position.
- The input pointer starts at column 1 of the first record, unless otherwise directed. As the INPUT statement executes, the raw data in columns 1-13 is read and is assigned to Item in the program data vector.

```
data perm.update;
  infile invent;
  input Item $ 1-13 IDnum $ 15-19
        InStock 21-22 BackOrd 24-25;
  Total=instock+backord;
run;
```



Raw Data File Invent

```
1---+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```

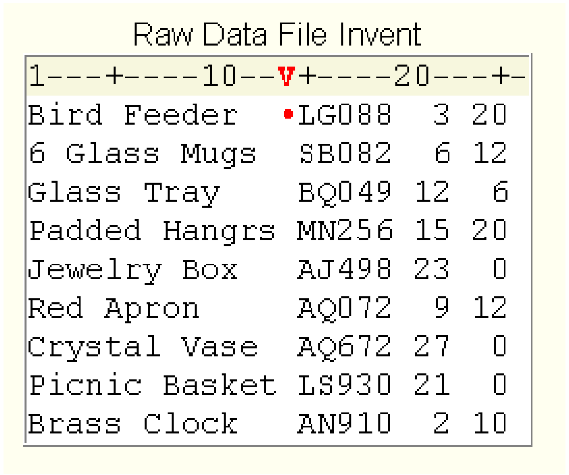
Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder		•	•	•

Execution Phase (6)

Input pointer (2)

- Notice that the input pointer now rests on column 14. With column input, the pointer moves as far as the INPUT statement instructs it, and it stops in the column immediately following the last one read.



Raw Data File Invent

```
1---+----10--V+----20---+-
Bird Feeder  •LG088   3  20
6 Glass Mugs  SB082   6  12
Glass Tray    BQ049  12   6
Padded Hangrs MN256  15  20
Jewelry Box   AJ498  23   0
Red Apron     AQ072   9  12
Crystal Vase  AQ672  27   0
Picnic Basket LS930  21   0
Brass Clock   AN910   2  10
```

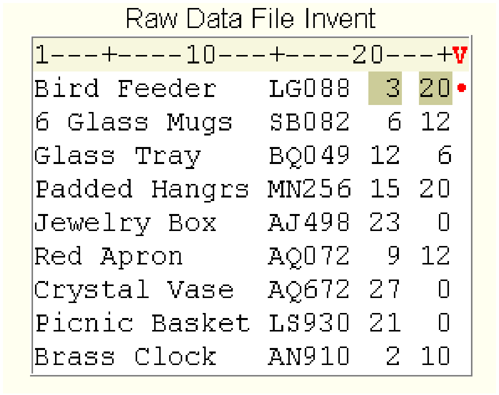
Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder		•	•	•

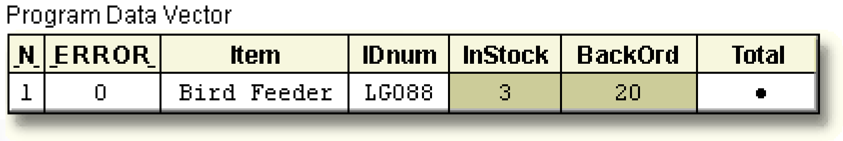
Execution Phase (7)

Input pointer (3)

- Next, the data in columns 15–19 is read and is assigned to IDnum in the program data vector, as shown below. Likewise, the INPUT statement reads the values for InStock from columns 21–22, and it reads the values for BackOrd from columns 24–25. At the end of the INPUT statement processing, the input pointer is in column 26.


Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088  3 20•
6 Glass Mugs   SB082  6 12
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```


Program Data Vector

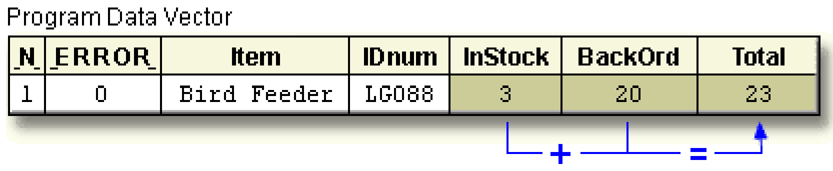
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	3	20	•

Execution Phase (8)

Input pointer (4)

- Next, the assignment statement executes. The values for InStock and BackOrd are added to produce the values for Total.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```



Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0	Bird Feeder	LG088	3	20	23

Execution Phase (9)

End of the DATA step (1)

- At the end of the DATA step, three actions occur. First, the values in the program data vector are written to the output data set as the first observation.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

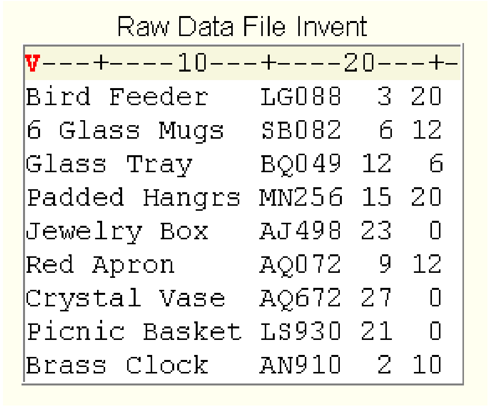
SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23

Execution Phase (10)

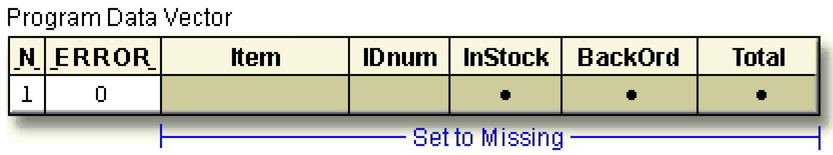
- End of the DATA step (2)
  - ➤ Next, the value of _N_ is set to 2 and control returns to the top of the DATA step. Finally, the variable values in the program data vector are re-set to missing. Notice that the automatic variable _ERROR_ retains its value.

```
data perm.update;
  infile invent;
  input Item $ 1-13 IDnum $ 15-19
        InStock 21-22 BackOrd 24-25;
  Total=instock+backord;
run;
```


Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```

  - ➤ Finally, the variable values in the program data vector are re-set to missing. Notice that the automatic variables _N_ and _ERROR_ retain their values.


Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
1	0			•	•	•

Set to Missing

Execution Phase (11)

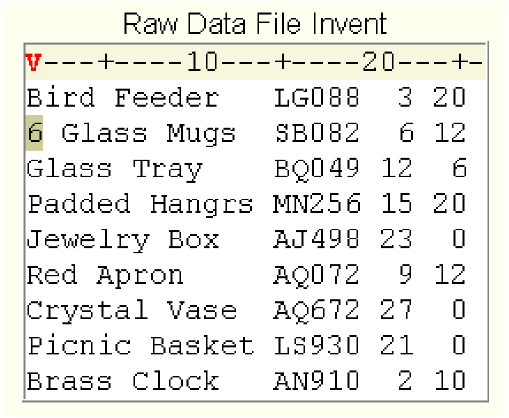
- End of the DATA step (3)
  - Note: When reading variables from **raw data**, SAS sets the value of each variable in the DATA step to missing at the beginning of each cycle of execution, with these exceptions:
    - variables that are named in a RETAIN statement
    - variables that are created in a sum statement
    - data elements in a _TEMPORARY_ array
    - any variables that are created with options in the FILE or INFILE statements
    - automatic variables.
  - In contrast, when reading variables from a SAS data set, SAS sets the values to missing only before the first cycle of execution of the DATA step. Thereafter, the variables retain their values until new values become available—for example, through an assignment statement or through the next execution of a SET or MERGE statement. Variables that are created with options in the SET or MERGE statements also retain their values from one cycle of execution to the next.

Execution Phase (12)

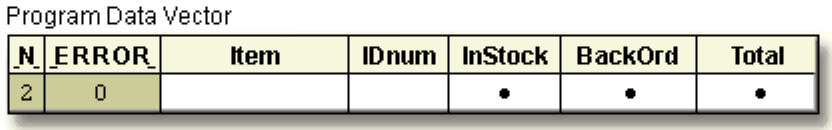
Iterations of the DATA step (1)

- You can see that the DATA step works like a loop, repetitively executing statements to read data values and create observations one by one. Each loop (or cycle of execution) is called an iteration. At the beginning of the second iteration, the value of _N_ is set to *2*, and _ERROR_ is still *0*. Notice that the input pointer rests in column 1 of the second record.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```


Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```


Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0			•	•	•

Execution Phase (13)

Iterations of the DATA step (2)

- As the INPUT statement executes for the second time, the values from the second record are held in the input buffer and then read into the program data vector.

```
data perm.update;
  infile invent;
  input Item $ 1-13 IDnum $ 15-19
        InStock 21-22 BackOrd 24-25;
  Total=instock+backord;
run;
```

Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12•
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```

Program Data Vector

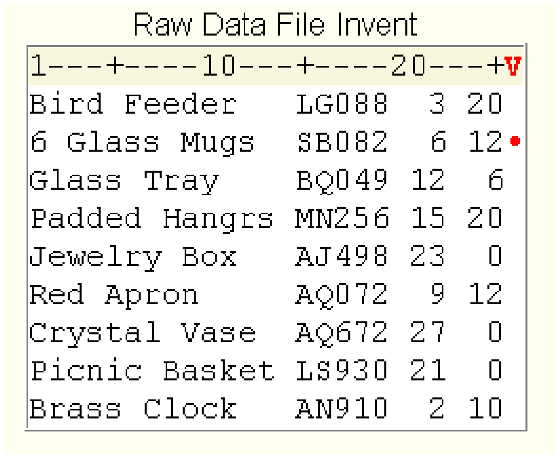
_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	

Execution Phase (14)

Iterations of the DATA step (3)

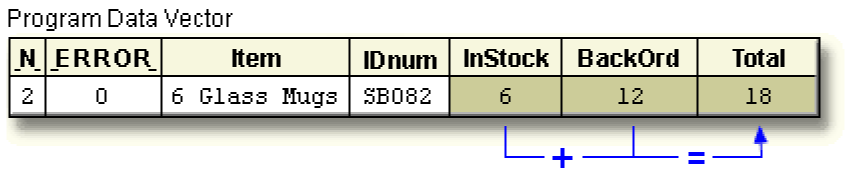
- Next, the value for Total is calculated based on the current values for InStock and BackOrd. The RUN statement indicates the end of the DATA step.

```
data perm.update;
   infile invent;
   input Item $ 1-13 IDnum $ 15-19
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```



Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12•
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```



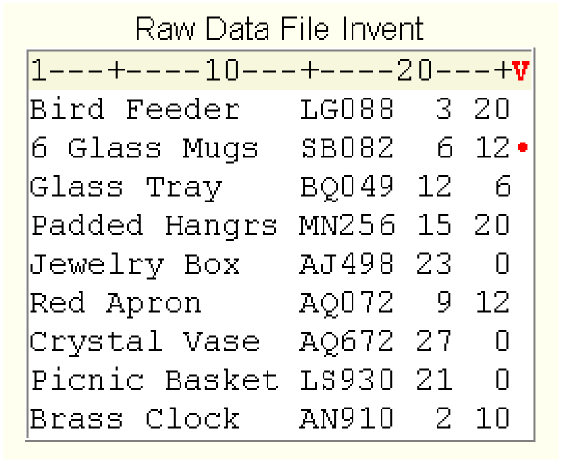
Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	18

Execution Phase (15)

Iterations of the DATA step (4)

➤ At the bottom of the DATA step, the values in the program data vector are written to the data set as the second observation.



Raw Data File Invent

```
1---+----10---+----20---+V
Bird Feeder    LG088  3 20
6 Glass Mugs   SB082  6 12•
Glass Tray     BQ049 12  6
Padded Hangrs  MN256 15 20
Jewelry Box    AJ498 23  0
Red Apron      AQ072  9 12
Crystal Vase   AQ672 27  0
Picnic Basket  LS930 21  0
Brass Clock    AN910  2 10
```

Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0	6 Glass Mugs	SB082	6	12	18

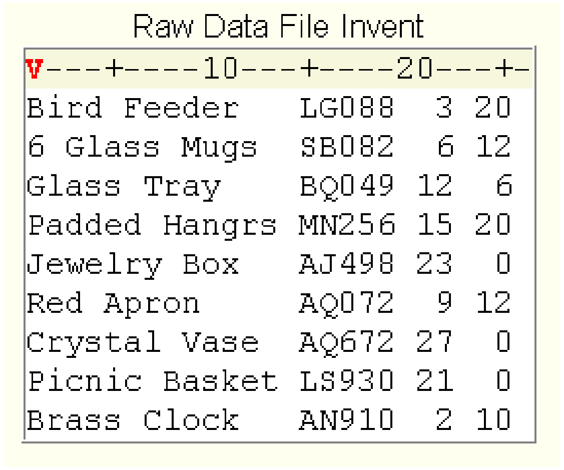
SAS Data Set Perm.Update

Item	IDnum	InStock	BackOrd	Total
Bird Feeder	LG088	3	20	23
6 Glass Mugs	SB082	6	12	18

Execution Phase (16)

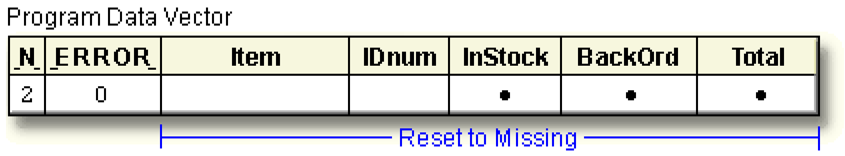
Iterations of the DATA step (5)

- Next, control returns to the top of the DATA step, and the values for Item, IDnum, InStock, BackOrd, and Total are re-set to missing.



Raw Data File Invent

```
V---+----10---+----20---+-
Bird Feeder    LG088   3  20
6 Glass Mugs   SB082   6  12
Glass Tray     BQ049  12   6
Padded Hangrs  MN256  15  20
Jewelry Box    AJ498  23   0
Red Apron      AQ072   9  12
Crystal Vase   AQ672  27   0
Picnic Basket  LS930  21   0
Brass Clock    AN910   2  10
```



Program Data Vector

_N_	_ERROR_	Item	IDnum	InStock	BackOrd	Total
2	0			•	•	•

Reset to Missing

Execution Phase (17)

End-of-file marker

- The execution phase continues in this manner until the end-of-file marker is reached in the raw data file. When there are no more records in the raw data file to be read, the data portion of the new data set is complete.
- Remember, the order in which variables are defined in the DATA step determines the order in which the variables are stored in the data set. The DATA step below, which reverses the order of Item and IDnum, produces a different data set from the same raw data.

```
data perm.update;
   infile invent;
   input IDnum $ 15-19 Item $ 1-13
         InStock 21-22 BackOrd 24-25;
   Total=instock+backord;
run;
```

SAS Data Set Perm.Update

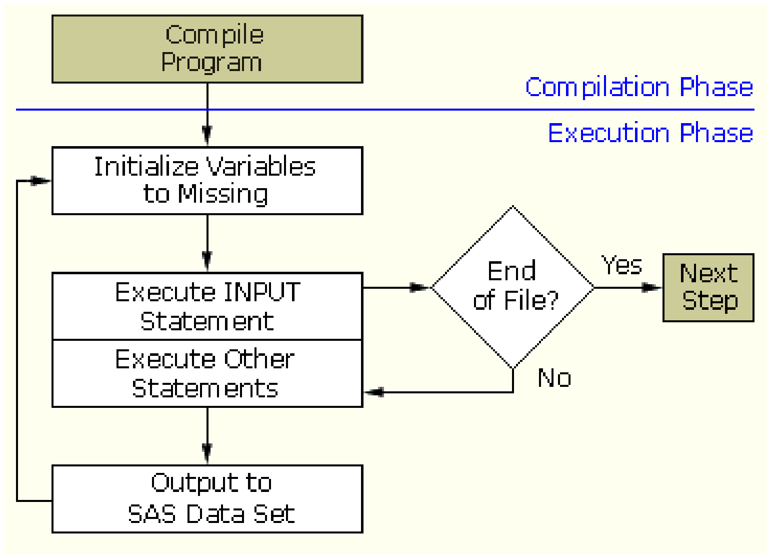
IDnum	Item	InStock	BackOrd	Total
LG088	Bird Feeder	3	20	23
SB082	6 Glass Mugs	6	12	18

Execution Phase (18)

- Summary of the execution phase (1)
  - You've seen how the DATA step iteratively reads records in the raw data file. Now take a minute to review execution-phase processing.
  - During the execution phase
    - variables in the program data vector are initialized to missing before each execution of the DATA step
    - each statement is executed sequentially
    - the INPUT statement reads the next record from the external file identified by the INFILE statement, and it writes the values into the program data vector
    - other statements can then further modify the current observation
    - the values in the program data vector are written to the SAS data set at the end of the DATA step
    - program flow is returned to the top of the DATA step
    - the DATA step is executed until the end-of-file marker is reached in the external file.

Execution Phase (19)

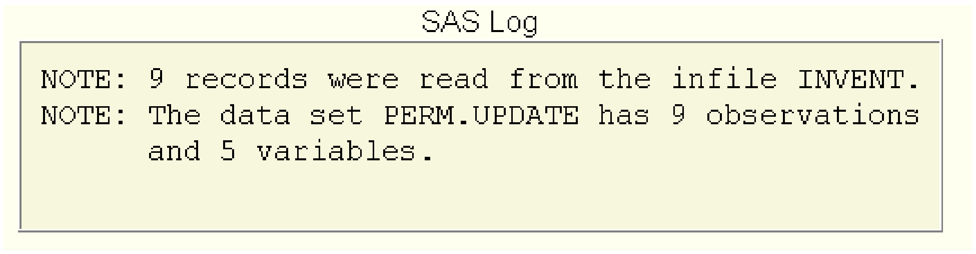
- Summary of the execution phase (2)



Execution Phase (20)

End of the execution phase

- At the end of the execution phase, the SAS log confirms that the raw data file was read, and it displays the number of observations and variables in the data set.


SAS Log

```
NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

Execution Phase (21)

- Unless otherwise directed, the DATA step executes
  - a. once for each compilation phase.
  - b. once for each DATA step statement.
  - c. once for each record in the input file.
  - d. once for each variable in the input file.

  - Correct answer: c
    - The DATA step executes once for each record in the input file, unless otherwise directed.

Execution Phase (22)

- Which of the following actions occurs at the end of the DATA step?

  - a. The automatic variables _N_ and _ERROR_ are incremented by one.
  - b. The DATA step stops execution.
  - c. The descriptor portion of the data set is written.
  - d. The values of variables created in programming statements are re-set to missing in the program data vector.

  - Correct answer: d
    - By default, at the end of the DATA step, the values in the program data vector are written to the data set as an observation, the value of the automatic variable _N_ is incremented by one, control returns to the top of the DATA step, and the values of variables created in programming statements are set to missing. The automatic variable _ERROR_ retains its value.

Debugging a DATA Step (1)

Diagnosing errors in the compilation phase (1)

- Now that you know how a DATA step is processed, you can use that knowledge to correct errors. Many errors are detected during the compilation phase, including
  - misspelled keywords and data set names
  - missing semicolons
  - unbalanced quotation marks
  - invalid options.

Debugging a DATA Step (2)

Diagnosing errors in the compilation phase (2)

- During the compilation phase, SAS can interpret some syntax errors (such as the keyword DATA misspelled as DAAT). If it cannot interpret the error, SAS
  - prints the word ERROR followed by an error message in the log
  - compiles but does not execute the step where the error occurred, and prints the following message to warn you:
    - NOTE: The SAS System stopped processing this step because of errors.
- Some errors are explained fully by the message that SAS prints; other error messages are not as easy to interpret. For example, because SAS statements are in free format, when you fail to end a SAS statement with a semicolon, SAS does not always detect the error at the point where it occurs.

Debugging a DATA Step (3)

Diagnosing errors in the execution phase (1)

As you have seen, errors can occur in the compilation phase, resulting in a DATA step that is compiled but not executed. Errors can also occur during the execution phase. When SAS detects an error in the execution phase, the following can occur, depending on the type of error:

- A note, warning, or error message is displayed in the log.
- The values that are stored in the program data vector are displayed in the log.
- The processing of the step either continues or stops.

Debugging a DATA Step (4)

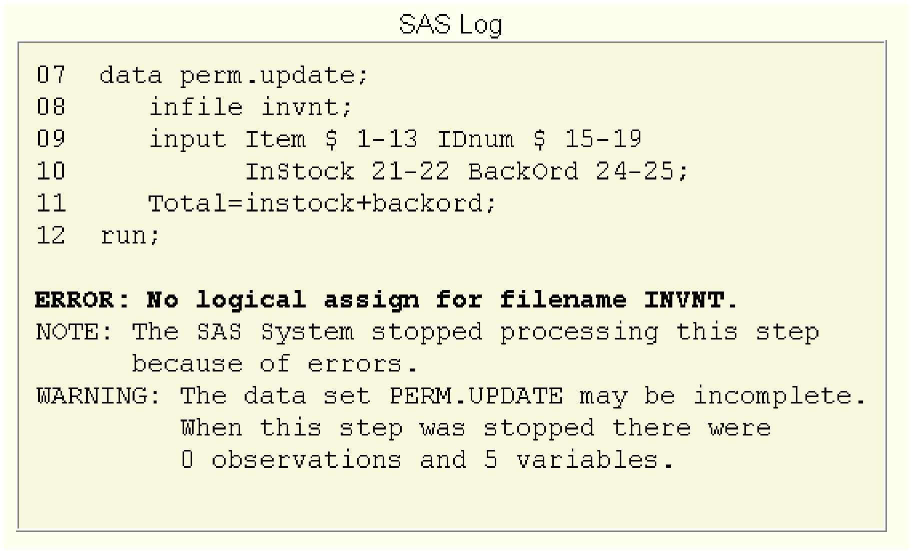
Diagnosing errors in the execution phase (2)

- Example: Suppose you misspelled the fileref in the INFILE statement below. This is not a syntax error, because SAS does not validate the file that you reference until the execution phase. During the compilation phase, the fileref **Invnt** is assumed to reference some external raw data file.

```
data perm.update;
     infile invnt;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

Debugging a DATA Step (5)

- Diagnosing errors in the execution phase (3)
  - This error is not detected until the execution phase begins. Because there is no external file that is referenced by the fileref **Invnt**, the DATA step stops processing.


SAS Log

```
07   data perm.update;
08       infile invnt;
09       input Item $ 1-13 IDnum $ 15-19
10               InStock 21-22 BackOrd 24-25;
11       Total=instock+backord;
12   run;

ERROR: No logical assign for filename INVNT.
NOTE: The SAS System stopped processing this step
      because of errors.
WARNING: The data set PERM.UPDATE may be incomplete.
         When this step was stopped there were
         0 observations and 5 variables.
```

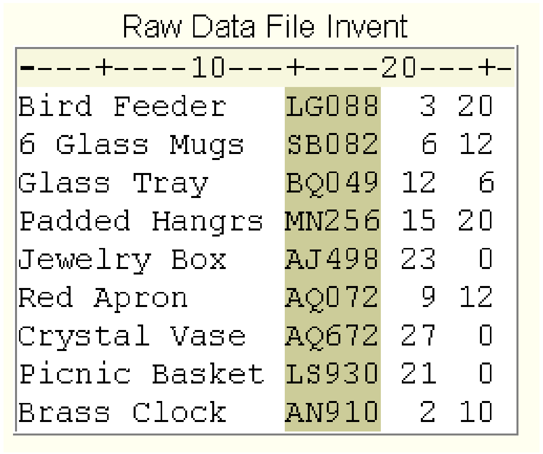
  - Because **Invent** is misspelled, the statement in the DATA step that identifies the raw data is incorrect. Note, however, that the correct number of variables was defined in the descriptor portion of the data set.

Debugging a DATA Step (6)

Diagnosing errors in the execution phase (4)

- Incorrectly identifying a variable's type is another common execution-time error. As you know, the values for IDnum are character values. Suppose you forget to place the dollar sign ($) after the variable's name in your INPUT statement. This is not a compile-time error, because SAS cannot verify IDnum's type until the data values for IDnum are read.

```
data perm.update;
     infile invnt;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

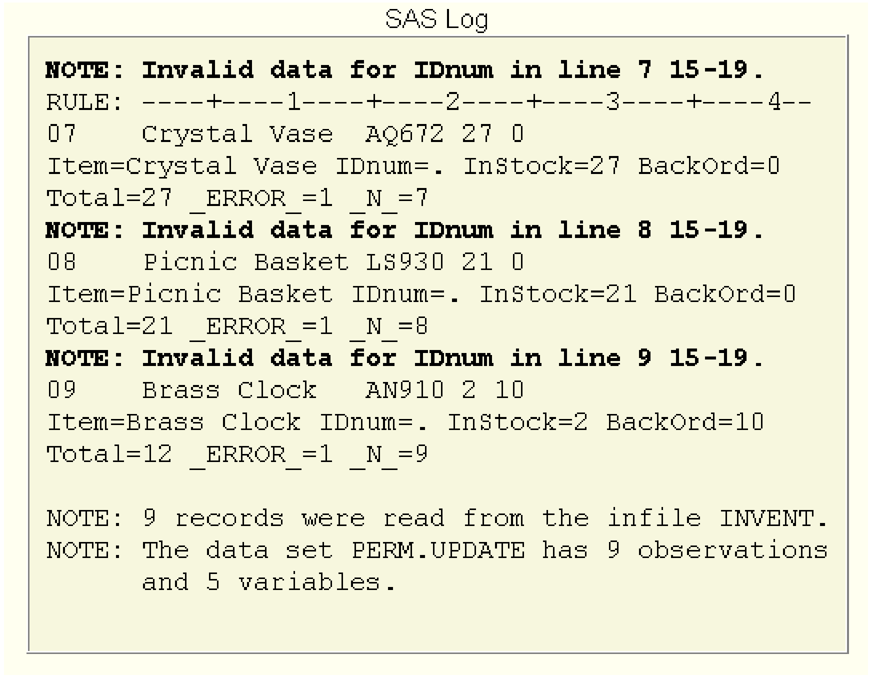

Raw Data File Invent

```
----+----10---+----20---+-
Bird Feeder   LG088  3 20
6 Glass Mugs  SB082  6 12
Glass Tray    BQ049 12  6
Padded Hangrs MN256 15 20
Jewelry Box   AJ498 23  0
Red Apron     AQ072  9 12
Crystal Vase  AQ672 27  0
Picnic Basket LS930 21  0
Brass Clock   AN910  2 10
```

Debugging a DATA Step (7)

Diagnosing errors in the execution phase (5)

- In this case, the DATA step completes the execution phase, and the observations are written to the data set. However, several notes appear in the log.


SAS Log

```
NOTE: Invalid data for IDnum in line 7 15-19.
RULE: ----+----1----+----2----+----3----+----4--
07    Crystal Vase  AQ672 27 0
Item=Crystal Vase IDnum=. InStock=27 BackOrd=0
Total=27 _ERROR_=1 _N_=7
NOTE: Invalid data for IDnum in line 8 15-19.
08    Picnic Basket LS930 21 0
Item=Picnic Basket IDnum=. InStock=21 BackOrd=0
Total=21 _ERROR_=1 _N_=8
NOTE: Invalid data for IDnum in line 9 15-19.
09    Brass Clock   AN910 2 10
Item=Brass Clock IDnum=. InStock=2 BackOrd=10
Total=12 _ERROR_=1 _N_=9

NOTE: 9 records were read from the infile INVENT.
NOTE: The data set PERM.UPDATE has 9 observations
      and 5 variables.
```

- Each note identifies the location of the invalid data for each observation. In this example, the invalid data is located in columns 15-19 for all observations.

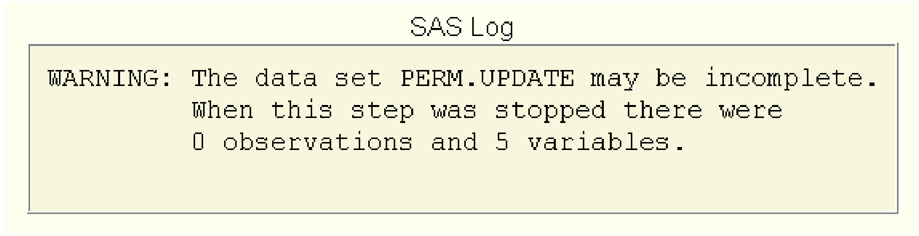
Debugging a DATA Step (8)

- Diagnosing errors in the execution phase (6)
  - The second line in each note (excluding the RULE line) displays the raw data record. Notice that the second field displays the values for IDnum, which are obviously character values.
  - The third and fourth lines display the values that are stored in the program data vector. Here, the values for IDnum are missing, although the other values have been correctly assigned to their respective variables. Notice that _ERROR_ has a value of *1*, indicating that an error has occurred.

Debugging a DATA Step (9)

Diagnosing errors in the execution phase (7)

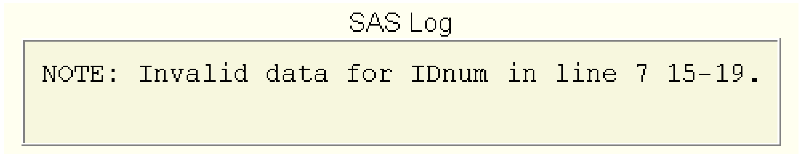
- When you read raw data with the DATA step, it's important to check the SAS log to verify that your data was read correctly. Here is a typical message:



SAS Log

```
WARNING: The data set PERM.UPDATE may be incomplete.
         When this step was stopped there were
         0 observations and 5 variables.
```

- When no observations are written to the data set, you should check to see whether your DATA step was completely executed. Most likely, a syntax error or another error is being detected at the beginning of the execution phase.



SAS Log

```
NOTE: Invalid data for IDnum in line 7 15-19.
```

- An **invalid data** message indicates that the program executed, but the data is not acceptable. Typically, the message indicates that a variable's type has been incorrectly identified in the INPUT statement, or that the raw data file contains some invalid data value(s).

Testing Your Programs (1)

- Writing a NULL data set

  ► After you write or edit a DATA step, you can compile and execute your program without creating observations. This enables you to detect the most common errors and saves you development time. A simple way to test a DATA step is to specify the keyword **_NULL_** as the data set name in the DATA statement.

```
data _null_;
     infile invent;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

  ► When you submit the DATA step, no data set is created, but any compilation or execution errors are written to the log. After correcting any errors, you can replace _NULL_ with the name of the data set you want to create.

Testing Your Programs (2)

Limiting observations

- Remember that you can use the **OBS=** option in the INFILE statement to limit the number of observations that are read or created during the execution of the DATA step.

```
data perm.update;
     infile invent obs=10;
     input Item $ 1-13 IDnum $ 15-19
           InStock 21-22 BackOrd 24-25;
     Total=instock+backord;
run;
```

- When processed, this DATA step creates the **Perm.Update** data set with variables but with only ten observations.

Testing Your Programs (3)

PUT statement (1)

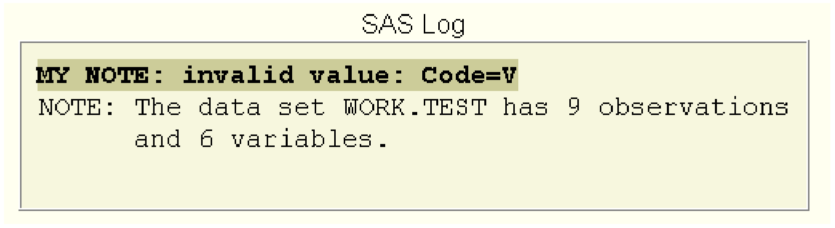
➤ When the source of program errors is not apparent, you can use the **PUT statement** to examine variable values and to print your own message in the log. For diagnostic purposes, you can use IF-THEN/ELSE statements to conditionally check for values.

```
data work.test;
     infile loan;
     input Code $ 1 Amount 3-10 Rate 12-16
           Account $ 18-25 Months 27-28;
     if code='1' then type='variable';
     else if code='2' then type='fixed';
     else put 'MY NOTE: invalid value: ' code=;
run;
```

Testing Your Programs (4)

PUT statement (2)

- In this example, if CODE does not have the expected values of *1* or *2*, the PUT statement writes a message to the log:



SAS Log

```
MY NOTE: invalid value: Code=V
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

- General form, simple PUT statement:

PUT *specification(s)*;

where *specification* specifies what is written, how it is written, and where it is written. This can include

  - a character string
  - one or more data set variables
  - the automatic variables _N_ and _ERROR_
  - the automatic variable _ALL_

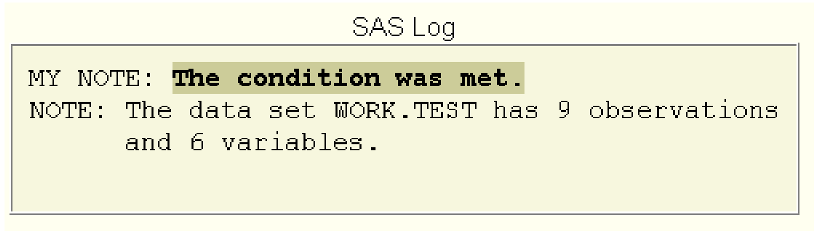
- and much more. The following pages show examples of PUT specifications.

Testing Your Programs (5)

Character strings

- You can use a PUT statement to specify a character string to identify your message in the log. The character string must be enclosed in quotation marks.

```
put 'MY NOTE: The condition was met.';
```

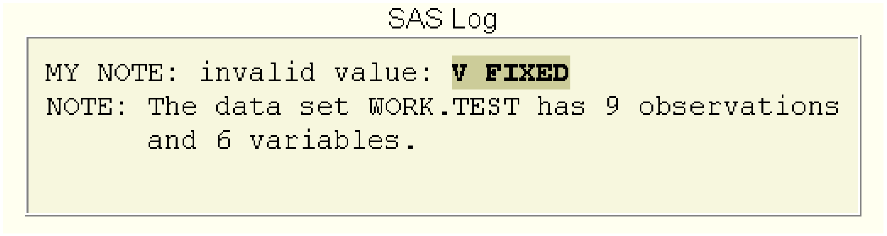


```
MY NOTE: The condition was met.
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

Testing Your Programs (6)

- Data set variables
  - You can use a PUT statement to specify one or more data set variables to be examined for that iteration of the DATA step:

```
put 'MY NOTE: invalid value: ' code type;
```

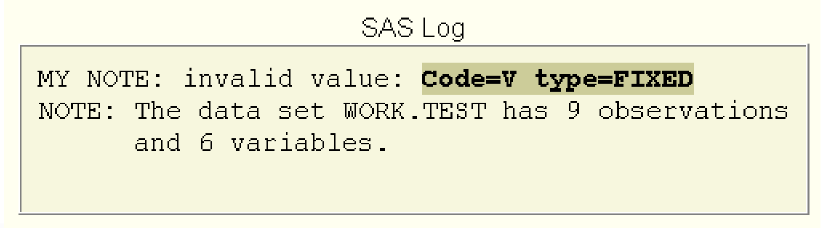


SAS Log

```
MY NOTE: invalid value: V FIXED
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

  - Note that when you specify a variable in the PUT statement, only its value is written to the log. To write both the variable name and its value in the log, add an equal sign (=) to the variable name.

```
put 'MY NOTE: invalid value: ' code= type=;
```



SAS Log

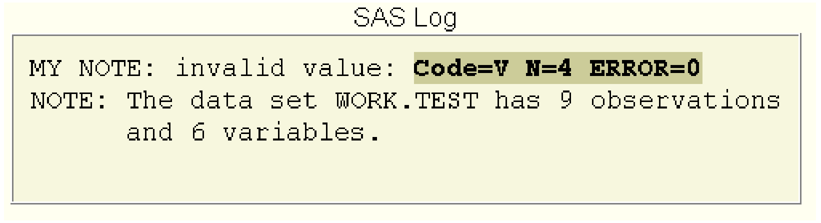
```
MY NOTE: invalid value: Code=V type=FIXED
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

Testing Your Programs (7)

Automatic variables

► You can use a PUT statement to display the values of the automatic variables _N_ and _ERROR_. In some cases, knowing the value of _N_ can help you locate an observation in the data set:

```
put 'MY NOTE: invalid value: '
     code= _n_= _error_=;
```

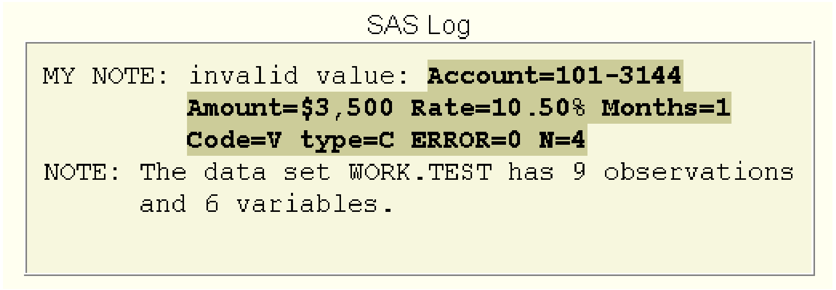


SAS Log

```
MY NOTE: invalid value: Code=V N=4 ERROR=0
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

► You can also use a PUT statement to write all variable names and variable values, including automatic variables, to the log. Use the _ALL_ specification:

```
put 'MY NOTE: invalid value:
     ' _all_ ;
```



SAS Log

```
MY NOTE: invalid value: Account=101-3144
        Amount=$3,500 Rate=10.50% Months=1
        Code=V type=C ERROR=0 N=4
NOTE: The data set WORK.TEST has 9 observations
      and 6 variables.
```

Testing Your Programs (8)

Conditional processing

- You can use a PUT statement with conditional processing (that is, with IF-THEN/ELSE statements) to flag program errors or data that is out of range. In the example below, the PUT statement is used to flag any missing or zero values for the variable Rate.

```
data finance.newcalc;
        infile newloans;
        input LoanID $ 1-4 Rate 5-8 Amount 9-19;
        if rate>0 then Interest=amount*(rate/12);
        else put 'DATA ERROR ' rate= _n_=;
run;
```

- The PUT statement can accomplish a wide variety of tasks. This lesson shows a few ways to use the PUT statement to help you debug a program or examine variable values.