Paper 73-26

That Mysterious Colon (:)

Haiping Luo, Dept. of Veterans Affairs, Washington, DC

ABSTRACT

The colon (:) plays certain roles in SAS coding. Its usage, however, is not well documented nor is it clearly indexed in SAS manuals. This paper shows how a colon can be used as a label indicator, an operator modifier, a format modifier, a key word component, a variable name wildcard, an array bound delimiter, an argument feature delimiter, a special log indicator, or an index creation operator. Mastering these usages can give your code needed functionality and/or an efficiency lift.

AN OVERVIEW

In SAS language, the colon (:) has many different uses, although they are not well documented. It is difficult to search for the colon's usage in SAS OnlineDoc, System Help, and printed manuals. From the scattered documentations, publications, and featured programmers, this paper collected nine types of colon usages:

1. Label indicator
2. Format modifier
3. Operator modifier
4. Key word component
5. Variable name wildcard
6. Array bound delimiter
7. Argument feature delimiter
8. Special log indicator
9. Index creation operator

Some of these usages can improve coding efficiency while others provide unique and necessary capacities for various circumstances. This paper will use examples to demonstrate different ways in which the colon can be used. Most of the examples in this paper were tested under SAS for Windows Version 8. Some of them were tested under version 6.12 and found not working. These known invalid cases are indicated in the text.

1. LABEL INDICATOR

A colon after a string signals the string as a label and the statement(s) after the colon as the labeled statement(s). The label string must be a valid SAS name. Any statement(s) within a data step can be labeled, although no two labels in a data step can have the same name. The statement label identifies the destination of a GO TO statement, a LINK statement, the HEADER= option in a FILE statement, or the EOF= option in an INFILE statement.

The labeled statement referred to by a GO TO or a LINK statement is used to alter the sequential flow of program execution. For example, in the following code:

```
data a;
 input x y z;
 if x=y then go to yes;
 x=10;
 y=32;
 return;
yes:
 put x= z=;
 delete;
 cards;
 . . .
```

statements 'x=10;' 'y=32;' will be executed for all observations when x is not equal to y. The statement 'return;' brings the execution back to the beginning of the data step for the next observation, without executing the two statements after the 'yes:' label. Only when the condition 'x=y' is met, will the program jump to the label 'yes:' and execute the two statements which follow the label. The value of x and z will be printed to the log, the observation will be deleted and the then the program will read in the next observation.

Similarly, a LINK statement also branches execution to statements after a label. The difference between the GOTO and the LINK statements is that after the execution the code following the label (the labeled statement group), a 'return;' statement in a LINK structure will bring execution to the statement following the LINK statement, while a 'return;' in a GOTO structure will bring execution to the beginning of the data step. The use of label in a LINK structure can be seen in the following example:

```
data workers;
 set tickets; by ssn;
 if first.ssn then link init;
 tickets+1;
 tothrs+hours;
 if last.ssn then output;
 return;
init:
 tickets=0;
 tothrs=0;
 return;
```

This data step sums tickets and total hours for each worker in the dataset 'tickets' and outputs the sums to dataset 'workers'. The statements after the label 'init' are only executed for the first observation of a social security number. After execution of the labeled statement group, execution continues to 'tickets+1' for the same first.ssn observation since this is the statement immediately following the LINK statement. If GO To were used instead of LINK, execution would cause immediate reading of the next observation without incrementing 'tickets' and 'tothrs' with data from the first observation for this worker.

Label can be used in report writing to execute statements when a new report page is begun. The following code uses a data step to write a customized report with a HEADER=*label* option in a file statement:

```
data _null_;
 set sales;
 by dept;
 /* header refers to the statements after the
label newpage */
 file print header=newpage;
 /*Start a new page for each department:*/
 if first.dept then put _page_;
  put @22 salesrep @34 salesamt;
  return;
 /* the put statement is executed for each
new page */
newpage:
 put @20 'Sales for 2000' /
     @20 dept=;
 return;
run;
```

Label can also be used in the EOF=*label* option of an INFILE statement to indicate how execution is to proceed after the end of

a file is reached. In the following code, suppose mydat.txt has 3 observations while mydat2.txt has 6. Without EOF=, the input statement will stop reading at observation 3 when it reaches the end of mydat.txt; dataset a will have only 3 observations and 6 variables. With EOF=more and the label 'more:', the data step continues, so the remaining 3 observations in mydat2.txt are read by the second input statement: dataset a will have 6 observations and 6 variables.

```
data a;
 infile "d:\mydat.txt" eof=more ;
 input @1 name $20. @21 x 5.1 @26 y 8.;
more:
 infile "d:\mydat2.txt";
 input @1 book $20. @21 test 8.1 @29 st 8.;
run;
```

In SAS Macro Language, there is a %GOTO statement that branches macro execution to a labeled section within the same macro. Branching with the %GOTO statement has two restrictions. First, the label that is the target of the %GOTO statement must exist in the current macro. Second, a %GOTO statement cannot cause execution to branch to a point inside an iterative %DO, %DO %UNTIL, or %DO %WHILE loop that is not currently executing. The following example uses %GOTO to exit the macro when a specific type of error occurs:

```
%macro check(parm);
   %local status;
   %if &parm= %then %do;
     %put ERROR: You must supply a
       parameter to macro CHECK.;
     %goto exit;
   %end;
   more macro statements that test for
     error conditions . . .
   %if &status > 0 %then %do;
       %put ERROR: File is empty.;
       %goto exit;
   %end;
   more macro statements . . .
   %put Check completed successfully.;
%exit:
%mend check;
```

In SAS/AF, the colon is used as a section label indicator in label-return structures to group code for frame labels or other frame entries. The following sections define the code for a 'RUN' and a 'PRINT' button in a frame:

```
RUN:
  if levelte='_Required_' then do;
   _msg_='You must select a level before
running the report!!';
   return;
  end;
  _msg_='Please wait while your request is
processed.';
  refresh;
  call display('means.scl',name,levelte);
return;

PRINT:
  rc=woutput('print',
     'sasuser.profile.default');
  rc=woutput('clear');
return;
```

2. FORMAT MODIFIER

The colon as an input/output modifier is documented in SAS manuals. You can find examples of its use if you know to search for 'format/informat modifier' instead of 'colon'; alternatively, you find examples by getting lucky. For input, colon enables you to use an informat for reading a data value in an otherwise list input process. The colon informat modifier indicates that the value is to be read from the next nonblank column until the pointer reaches the next blank column or the end of the data line, whichever comes first. Though the data step continues reading until it reaches the next blank column, it truncates the value of a character variable if the field is longer that its formatted length. If the length of the variable has not been previously defined, its value is read and stored with the informat length.

In the following case, the input data are not lined up neatly. Some of the problems are: the first data line does not start in column 1; there are single spaces and a semi-colon embedded in character values; the value of 'city' in the second data line starts in column 12, which is within the defined range of the first variable; and there are 2 or more spaces between variables. Given these features of the data, if no colon is used, the code will produce the output printed at the end of the code:

```
/* Without colon in the input statement*/
data a;
  input student $14. city & $30.;
cards4;
              Jim Smith   Washington DC; L.A.
Key Jones  Chicago
;;;;
proc print;
run;
```

The output missed the starting character of the variables

```
 OBS      STUDENT         CITY
  1                       Jim Smith
  2    Key Jones  Chi     cago
```

If, however, a colon is added to the input statement:

```
input student : $14. city & $30.;
```

The output becomes:

```
 OBS     STUDENT     CITY
  1        Jim       Smith
  2        Key       Jones
```

There is still a problem – the variable values have not been separated correctly. If an ampersand (&) is also added to the input statement:

```
input student : & $14. city & $30.;
```

The output becomes what we wanted:

```
OBS      STUDENT      CITY
 1      Jim Smith     Washington DC; L.A.
 2      Key Jones     Chicago
```

In the above example, the colon causes the input statement to read from the next non-blank character until the pointer reaches the next blank column. The ampersands(&) tell SAS that single blanks may be embedded within character values. The combination of colon and ampersand causes the input statement to read from the next non-blank character until the pointer reaches double blanks. Also note that, due to the semi-colon within the data, a CARDS4 statement and 4 semi-colons (;;;;) are used to indicate the beginning and the end of the data lines.

For output, a colon preceding a format in a PUT statement forms a 'Modified List Output'. Modified List Output generates different results compared to that from a Formatted Output. List output and formatted output use different methods to determine how far to move the pointer after a variable value is written. Modified list output writes the value, inserts a blank space, and moves the pointer to the next column. All leading and trailing blanks are

deleted, and each value is followed by a single blank. Formatted output moves the pointer the length of the format, even if the value does not fill that length. The pointer moves to the next column; an intervening blank is not inserted. The following DATA step uses modified list output to write each output line:

```
data _null_;
   input x y;
   put x : comma10.2 y : 7.2;
   datalines;
2353.20 7.10
231 21
;
```

These lines are written to the SAS log, with the values separated by an inserted blank:

```
----+----1----+----2
2,353.20 7.10
231.00 21.00
```

In comparison, the following example uses formatted output:

```
put x comma10.2 y 7.2;
```

These lines are written to the SAS log, with the values aligned in columns:

```
----+----1----+----2
  2,353.20   7.10
    231.00 121.00
```

3. OPERATOR MODIFIER

In SAS, character strings must be adjusted to the same length before they can be compared. When SAS compares character strings without the colon modifier, it pads the shorter string with blanks to the length of the longer string before making the comparison. When SAS compares character strings with the colon modifier after the operator, it truncates the longer string to the length of the shorter string. This feature of the colon modifier makes the comparison of a character string's prefix possible. For example,

```
if zip='010' then do;
* This will pick up any zip which equals
'010' exactly;
if zip=:'010' then do;
* will pick up any zip starting with '010',
such as '01025','0103', '01098';
if zip>=:'010' then do;
* will pick up any zip from '010' up
alphabetically, such as '012', '21088';
where lastname gt: 'Sm';
* will pick up any last name alphabetically
higher than 'Sm', such as 'Smith','SNASH',
'Snash';
```

The colon modifier can follow all comparison operators (=:, >=:, <=:, ne:, gt:, lt:, in:) to conduct prefix comparison. The following 'in:' operation will select the students located in zip codes which begin with '010', '011', '0131', '0138', respectively:

```
data s; set student;
if zip in:('010','011','0131','0138');
```

4. KEY WORD COMPONENT

In PROC SQL, the colon is part of the "SELECT… INTO :" structure. This structure turns the values of a selected field into a string of a macro variable for later use. The syntax of this structure is:

```
SELECT field-name1, . . ., field-nameN
INTO :macro-variable-name1,. . .,
   :macro-variable-nameN,
[SEPARATED BY 'a blank, a character, or a
character string' [NOTRIM]]
FROM a dataset or table;
```

In the following example, we have a list of social security numbers (SSNs) to use as a basis for selecting matching records from a large, raw data file. There are a number of different ways to do this. A macro variable created via SELECT INTO: is an efficient way.

```
*** get base SSN list;
data ssn_list;
 input ssn $9.;
 datalines;
123456789
234567890
345678901
456789012
;
run;
*** create a macro variable ssnok;
proc sql noprint;
 select ssn into :ssnok
 separated by ', '
 from ssn_list;
quit;
*** display the value of the macro variable
in the LOG as
123456789, 234567890, 345678901, 456789012;
%put &ssnok;
*** dataset selected will contain only the
records of those SSNs in the base list;
data selected;
 input @1 ssn @;
*** use the macro variable &ssnok (a list of
SSNs) to filter records;
 if ssn not in (&ssnok) then return;
 input <list of variables>;
 output;
run;
```

Thanks to the macro variable &ssnok, the dataset 'selected' will have only those observations whose social security numbers match the SSNs in dataset 'ssn_list'. Note that there is a length limitation in this SELECT INTO: structure in case you are going to generate a very long list. The length of the generated macro variable cannot exceed 32K characters.

5. VARIABLE NAME WILDCARD

A colon following a variable name prefix selects any variable whose name starts with that prefix. This useful feature can make it possible for a short statement to process a large group of variables whose names begin with the same prefix. For example,

```
drop hi:;
*** This will drop any variable whose name
starts with 'hi', such as high, hi, hifi;
data b; set a(keep=d:);
*** This keeps every variables whose name
starts with d, such as d1, d2, degree, date;
total=sum(of times:);
*** The function sums all variables with the
prefix of 'times'. Note this last form is
invalid under version 6.12 or lower for
Windows;
```

6. ARRAY BOUND DELIMITER

We can use a method called array processing to perform the same tasks for a series of related variables within a dataset. The variables involved in the processing are related by arrays. An array is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array-name; its definition is valid only in the same data step.

By default in SAS, the subscript in each dimension of an array ranges from 1 to n, where n is the number of elements in that dimension. In an array statement, the lower bound can be omitted if its value is 1. An array with 2 dimensions ranging from 1 to 3 and 1 to 6, respectively, can be declared as:

```
array test{3,6} test1-test18;
```

Using a formal Bounded Array Declaration, we can declare this same array as:

```
array test{1:3,1:6} test1-test18;
```

In a bounded array declaration, the colon is used to separate the lower and upper bounds of an array's dimension. The bounded array declaration is useful in defining array dimensions which are not based on 1. In the following case, defining the array bounds according to year numbers makes the code more readable and reusable:

```
array sales{1980:2000} sales1980-sales2000;
do i=lbound(sales) to hbound(sales);
  . . .
   if sales{i}<0 then do;
     call symput('y',i);
     call symput('sales',sales{i});
     %put &y: Sales Error! Sales&y=&sales;
  end;
  . . .
end;
```

Suppose that 'sales1987' has a sales value of –5,000, that is, sales{1987}=-5000. The above code will put this line in the log (given that 'options symbolgen;' is in effect):

```
1987: Sales Error! Sales1987=-5000;
```

7. ARGUMENT FEATURE DELIMITER

In SAS/AF, a colon can be used both as an argument's feature delimiter and as a label indicator. In a METHOD statement, each argument type must be declared and colons are used to separate the elements of declaration:

```
label: method public
     arg1:output:num
     arg2:input:char(32)
endmethod;
```

In the argument segments of the above code, the first element, e.g. 'arg1', is the argument's name, the second, e.g., 'output', is the argument's usage, and the third, e.g., 'num', is the argument's data type.

8. SPECIAL LOG INDICATOR

In SAS for Windows version 8, a colon can be combined with keyword ERROR, NOTE, or WARNING in a %PUT statement to generate customized error, note, or warning text in SAS log. These particular user-defined log strings have the same color as the system generated error, note, and warning messages. In the %PUT statement, the keyword must be the first word after %PUT, must be in upper case, and must be followed immediately by a colon or a hyphen. The following statements

```
%put ERROR:  You made a wrong turn!;
%put NOTE: Your Lucky Number IS - &lucky.;
%put WARNING: Stop and Check!;
```

generate these colored strings in the log:

```
ERROR: You made a wrong turn! (Burgundy)
NOTE: Your Lucky Number IS - 9058. (Blue)
WARNING: Stop and Check! (Green)
```

9. INDEX CREATION OPERATOR

In SAS/IML(Interactive Matrix Language), a colon can serve as an index creation operator to create a row vector. The syntax for the vector creation is:

```
rowname = value1:valueN;
```

In the statement, the colon delimits the first element and the last element of the index row vector. When the first element of the vector is smaller than the last element, the elements in the row vector increment one by one from the left to the right. For example, the statement

```
normalrw = 4:9;
```

results in

```
normalrw  1 row   6 cols    (numeric)
4  5  6  7  8  9
```

When the first element of the vector is larger than the last element, a reverse order index is created with an incremental value of –1. This statement

```
reverserw = 10:3;
```

results in

```
reverserw  1 row   8 cols    (numeric)
10  9  8  7  6  5  4  3
```

When the elements of the vector are character arguments with a numeric suffix, the arguments should be put in quotation marks. The following statement

```
charrw = 'month3':'month7';
```

generates an index row of

```
charrw  1 row   5 cols    (character)
month3 month4 month5 month6 month7
```

If the element's increment is not 1 or -1, a DO function should replace the colon delimited phrase 'value1:value2' to generate the index row vector.

CONCLUSION

In SAS language, a colon (:) can be used in many different ways. This paper explained how it may be used as a label indicator, a format modifier, an operator modifier, a key word component, a variable name wildcard, an array bound delimiter, an argument feature delimiter, a special log indicator, and an index creation operator. Mastering these different uses can give you added flexibility in writing efficient, highly functional code.

REFERENCES

SAS Institute: *SAS OnlineDoc, version 8,* 1999

SAS Institute: *SAS System Help*, V6, V8, 1996, 1999

Grant, Paul: *SUGI 23: Simplifying Complex Character Comparisons by Using the IN Operator and the Colon (:) Operator Modifier*, 1998

Cody, Ronald P.: *SUGI 23: The INPUT Statement: Where It's @*, 1998

Kuligowski, Andrew T.; Roberts, Nancy: *SUGI 23: Basic Methods to Introduce External Data into the SAS System*, 1998

Timbers, Vincent L.: *SUGI 23: SAS/AF Frame Entries: A Hands-on Introduction*, 1998

Olson, Diane: *SUGI 25: Power Indexing: A Guide to Using Indexes Effectively In Nashville Releases*, 2000

Jaffe, Jay A.: *SUGI 24: SAS Macros: Beyond the Basics*, 1999

Satchi, Thiru: SUGI 24: An SQL List Macro to Retrieve Data from Large SAS/DB2 Databases, 1999

ACKNOWLEDGEMENT

I appreciate the contribution from Arthur L. Carpenter of California Occidental Consultants and Phillip Friend of USDA as well as my colleagues in the Department of Veterans Affairs.

CONTACT INFORMATION

Hope you can provide your cases of using colons in SAS. Please contact the author at:

Author Name	Haiping Luo
Email	hpluo@yahoo.com
SAS Discussion Board	http://clubs.yahoo.com/clubs/sas http://go.to/sas-net

---