Previous Page | Next Page

SAS Operators in Expressions

Definitions

A SAS **operator** is a symbol that represents a comparison, arithmetic calculation, or logical operation; a SAS function; or grouping parentheses. SAS uses two major types of operators:

- prefix operators
- infix operators.

A **prefix operator** is an operator that is applied to the variable, constant, function, or parenthetic expression that immediately follows it. The plus sign (+) and minus sign (-) can be used as prefix operators. The word NOT and its equivalent symbols are also prefix operators. The following are examples of prefix operators used with variables, constants, functions, and parenthetic expressions:

- `+y`
- `-25`
- `-cos(angle1)`
- `+(x*y)`

An **infix operator** applies to the operands on each side of it, for example, 6<8. Infix operators include the following:

- arithmetic
- comparison
- logical, or Boolean
- minimum
- maximum
- concatenation.

When used to perform arithmetic operations, the plus and minus signs are infix operators.

SAS also provides several other operators that are used only with certain SAS statements. The WHERE statement uses a special group of SAS operators, valid only when used with WHERE expressions. For a discussion of these operators, see WHERE-Expression Processing. The _NEW_ operator is used to create an instance of a DATA step component object. For more information, see Using DATA Step Component Objects.

Arithmetic Operators

Arithmetic operators indicate that an arithmetic calculation is performed, as shown in the following table:

Arithmetic Operators

Symbol	Definition	Example	Result
**	exponentiation	`a**3`	raise A to the third power
*	multiplication (table note 1)	`2*y`	multiply 2 by the value of Y
/	division	`var/5`	divide the value of VAR by 5
+	addition	`num+3`	add 3 to the value of NUM
-	subtraction	`sale-discount`	subtract the value of DISCOUNT from the value of SALE

TABLE NOTE 1: The asterisk (*) is always necessary to indicate multiplication; **`2Y`** and **`2(Y)`** are not valid expressions.

If a missing value is an operand for an arithmetic operator, the result is a missing value. See Missing Values for a discussion of how to prevent the propagation of missing values.

See Order of Evaluation in Compound Expressions for the order in which SAS evaluates these operators.

Comparison Operators

Comparison operators set up a comparison, operation, or calculation with two variables, constants, or expressions. If the comparison is true, the result is 1. If the comparison is false, the result is 0.

Comparison operators can be expressed as symbols or with their mnemonic equivalents, which are shown in the following table:

Comparison Operators

Symbol	Mnemonic Equivalent	Definition	Example
=	EQ	equal to	`a=3`
^=	NE	not equal to (table note 1)	`a ne 3`
¬=	NE	not equal to	
~=	NE	not equal to	
>	GT	greater than	`num>5`
<	LT	less than	`num<8`
>=	GE	greater than or equal to (table note 2)	`sales>=300`
<=	LE	less than or equal to (table	

		note 3)	`sales<=100`
	IN	equal to one of a list	`num in (3, 4, 5)`

TABLE NOTE 1: The symbol you use for NE depends on your personal computer.

TABLE NOTE 2: The symbol => is also accepted for compatibility with previous releases of SAS. It is not supported in WHERE clauses or in PROC SQL.

TABLE NOTE 3: The symbol =< is also accepted for compatibility with previous releases of SAS. It is not supported in WHERE clauses or in PROC SQL.

See Order of Evaluation in Compound Expressions for the order in which SAS evaluates these operators.

Note: You can add a colon (:) modifier to any of the operators to compare only a specified prefix of a character string. See Character Comparisons for details.

Note: You can use the IN operator to compare a value that is produced by an expression on the left of the operator to a list of values that are given on the right. The form of the comparison is:

```
expression IN (value-1<...,value-n>)
```

The components of the comparison are as follows:

expression can be any valid SAS expression, but is usually a variable name when it is used with the IN operator.

value must be a constant.

For examples of using the IN operator, see The IN Operator in Numeric Comparisons.

Numeric Comparisons

SAS makes numeric comparisons that are based on values. In the expression A<=B, if A has the value 4 and B has the value 3, then A<=B has the value 0, or false. If A is 5 and B is 9, then the expression has the value 1, or true. If A and B each have the value 47, then the expression is true and has the value 1.

Comparison operators appear frequently in IF-THEN statements, as in this example:

```
if x<y then c=5;
   else c=12;
```

You can also use comparisons in expressions in assignment statements. For example, the preceding statements can be recoded as follows:

```
c=5*(x<y)+12*(x>=y);
```

Since SAS evaluates quantities inside parentheses before performing any operations, the expressions **(x<y)** and **(x>=y)** are evaluated first and the result (1 or 0) is substituted for the expressions in parentheses. Therefore, if X=6 and Y=8, the expression evaluates as follows:

```
c=5*(1)+12*(0)
```

The result of this statement is C=5.

You might get an incorrect result when you compare numeric values of different lengths because values less than 8 bytes have less precision than those longer than 8 bytes. Rounding also affects the outcome of numeric comparisons. See SAS Variables for a complete discussion of numeric precision.

A missing numeric value is smaller than any other numeric value, and missing numeric values have their own sort order. See Missing Values for more information.

The IN Operator in Numeric Comparisons

You can use a shorthand notation to specify a range of sequential integers to search. The range is specified by using the syntax M:N as a value in the list to search, where M is the lower bound and N is the upper bound. M and N must be integers, and M, N, and all the integers between M and N are included in the range. For example, the following statements are equivalent.

- **y = x in (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);**
- **y = x in (1:10);**

You can use multiple ranges in the same IN list, and you can use ranges with other constants in an IN list. The following example shows a range that is used with other constants to test if X is 0, 1, 2, 3, 4, 5, or 9.

```
if x in (0,9,1:5);
```

You can also use the IN operator to search an array of numeric values. For example, the following code creates an array **a**, defines a constant **x**, and then uses the IN operator to search for **x** in array **a** Note that the array initialization syntax of **array a{10} (2*1:5)** creates an array that contains the initial values of 1, 2, 3, 4, 5, 1, 2, 3, 4, 5.

```
data _null_;
   array a{10} (2*1:5);
   x=99;
   y = x in a;
   put y=;
   a{5} = 99;
   y = x in a;
   put y=;
run;
```

Results From Using the IN Operator to Search an Array of Numeric Values (partial output)

```
1    data _null_;
2       array a[10] (2*1:5);
3       x=99;
4       y = x in a;
5       put y=;
6       a[5] = 99;
7       y = x in a;
8       put y=;
9    run;
y=0
y=1
```

Note: PROC SQL does not support this syntax. ■

Character Comparisons

You can perform comparisons on character operands, but the comparison always yields a numeric result (1 or 0). Character operands are compared character by character from left to right. Character order depends on the **collating**

sequence, usually ASCII or EBCDIC, used by your computer.

For example, in the EBCDIC and ASCII collating sequences, **G** is greater than **A** ; therefore, this expression is true:

```
'Gray'>'Adams'
```

Two character values of unequal length are compared as if blanks were attached to the end of the shorter value before the comparison is made. A blank, or missing character value, is smaller than any other printable character value. For example, because **.** is less than **h** , this expression is true:

```
'C. Jones'<'Charles Jones'
```

Since trailing blanks are ignored in a comparison, **'fox '** is equivalent to **'fox'** . However, because blanks at the beginning and in the middle of a character value are significant to SAS, **' fox'** is not equivalent to **'fox'** .

You can compare only a specified prefix of a character expression by using a colon (:) after the comparison operator. SAS truncates the longer value to the length of the shorter value during the comparison. In the following example, the colon modifier after the equal sign tells SAS to look at only the first character of values of the variable LASTNAME and to select the observations with names beginning with the letter **S** :

```
if lastname=:'S';
```

Because printable characters are greater than blanks, both of the following statements select observations with values of LASTNAME that are greater than or equal to the letter **S** :

- **if lastname>='S';**
- **if lastname>=:'S';**

Note: If you compare a zero-length character value with any other character value in either an IN: comparison or an EQ: comparison, the two character values are not considered equal. The result always evaluates to 0, or false. ■

The operations that are discussed in this section show you how to compare entire character strings and the beginnings of character strings. Several SAS character functions enable you to search for and extract values from within character strings. See **SAS Language Reference: Dictionary** for complete descriptions of all SAS functions.

The IN Operator in Character Comparisons

You can use the IN operator with character strings to determine whether a variable's value is among a list of character values. The following statements produce the same results:

- **if state in ('NY','NJ','PA') then region+1;**
- **if state='NY' or state='NJ' or state='PA' then region+1;**

You can also use the IN operator to search an array of character values. For example, the following code creates an array **a**, defines a constant **x**, and then uses the IN operator to search for **x** in array **a**.

```
data _null_;
   array a{5} $ (5*'');
   x='b1';
   y = x in a;
   put y=;
   a{5} = 'b1';
   y = x in a;
   put y=;
run;
```

Results From Using the IN Operator to Search an Array of Character Values (partial output)

```
1    data _null_;
2       array a{5} $ (5*'');
3       x='b1';
4       y = x in a;
5       put y=;
6       a{5} = 'b1';
7       y = x in a;
8       put y=;
9    run;
y=0
y=1
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Logical (Boolean) Operators and Expressions

Logical operators, also called **Boolean operators**, are usually used in expressions to link sequences of comparisons. The logical operators are shown in the following table:

Logical Operators

Symbol	Mnemonic Equivalent	Example
&	AND	`(a>b & c>d)`
|	OR (table note 1)	`(a>b or c>d)`
!	OR	
¦	OR	
¬	NOT (table note 2)	`not(a>b)`
^	NOT	
~	NOT	

TABLE NOTE 1: The symbol you use for OR depends on your operating environment.

TABLE NOTE 2: The symbol you use for NOT depends on your operating environment.

See Order of Evaluation in Compound Expressions for the order in which SAS evaluates these operators.

In addition, a numeric expression without any logical operators can serve as a Boolean expression. For an example of Boolean numeric expressions, see Boolean Numeric Expressions.

The AND Operator

If **both** of the quantities linked by AND are 1 (true), then the result of the AND operation is 1; otherwise, the result is 0. For example, in the following comparison:

```
a<b & c>0
```

the result is true (has a value of 1) only when both A<B **and** C>0 are 1 (true): that is, when A is less than B **and** C is positive.

Two comparisons with a common variable linked by AND can be condensed with an implied AND. For example, the following two subsetting IF statements produce the same result:

- **if 16<=age and age<=65;**
- **if 16<=age<=65;**

The OR Operator

If **either** of the quantities linked by an OR is 1 (true), then the result of the OR operation is 1 (true); otherwise, the OR operation produces a 0. For example, consider the following comparison:

```
a<b|c>0
```

The result is true (with a value of 1) when A<B is 1 (true) regardless of the value of C. It is also true when the value of C>0 is 1 (true), regardless of the values of A and B. Therefore, it is true when either or both of those relationships hold.

Be careful when using the OR operator with a series of comparisons (in an IF, SELECT, or WHERE statement, for example). Remember that only one comparison in a series of OR comparisons must be true to make a condition true, and any nonzero, nonmissing constant is always evaluated as true (see Boolean Numeric Expressions). Therefore, the following subsetting IF statement is always true:

```
if x=1 or 2;
```

SAS first evaluates X=1, and the result can be either true or false; however, since the 2 is evaluated as nonzero and nonmissing (true), the entire expression is true. In this statement, however, the condition is not necessarily true because either comparison can evaluate as true or false:

```
if x=1 or x=2;
```

The NOT Operator

The prefix operator NOT is also a logical operator. The result of putting NOT in front of a quantity whose value is 0 (false) is 1 (true). That is, the result of negating a false statement is 1 (true). For example, if X=Y is 0 (false) then NOT(X=Y) is 1 (true). The result of NOT in front of a quantity whose value is missing is also 1 (true). The result of NOT in front of a quantity with a nonzero, nonmissing value is 0 (false). That is, the result of negating a true statement is 0 (false).

For example, the following two expressions are equivalent:

- **not(name='SMITH')**
- **name ne 'SMITH'**

Furthermore, NOT(A&B) is equivalent to NOT A|NOT B, and NOT(A|B) is the same as NOT A & NOT B. For example, the following two expressions are equivalent:

- **not(a=b & c>d)**
- **a ne b | c le d**

Boolean Numeric Expressions

In computing terms, a value of true is a 1 and a value of false is a 0. In SAS, any numeric value other than 0 or missing is true, and a value of 0 or missing is false. Therefore, a numeric variable or expression can stand alone in a condition. If its value is a number other than 0 or missing, the condition is true; if its value is 0 or missing, the condition is false.

```
0 | . = False
1 = True
```

For example, suppose that you want to fill in variable REMARKS depending on whether the value of COST is present for a given observation. You can write the IF-THEN statement as follows:

```
if cost then remarks='Ready to budget';
```

This statement is equivalent to:

```
if cost ne . and cost ne 0
   then remarks='Ready to budget';
```

A numeric expression can be simply a numeric constant, as follows:

```
if 5 then do;
```

The numeric value that is returned by a function is also a valid numeric expression:

```
if index(address,'Avenue') then do;
```

The MIN and MAX Operators

The MIN and MAX operators are used to find the minimum or maximum value of two quantities. Surround the operators with the two quantities whose minimum or maximum value you want to know. The MIN (><) operator returns the lower of the two values. The MAX (<>) operator returns the higher of the two values. For example, if A<B, then A><B returns the value of A.

If missing values are part of the comparison, SAS uses the sorting order for missing values that is described in Order of Missing Values. For example, the maximum value that is returned by .A<>.Z is the value .Z.

Note: In a WHERE statement or clause, the <> operator is equivalent to NE. ■

The Concatenation Operator

The concatenation operator (||) concatenates character values. The results of a concatenation operation are usually stored in a variable with an assignment statement, as in **`level='grade '||'A'`** . The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation, unless you use a LENGTH or ATTRIB statement to specify a different length for the new variable.

The concatenation operator does not trim leading or trailing blanks. If variables are padded with trailing blanks, check the lengths of the variables and use the TRIM function to trim trailing blanks from values before concatenating them. See **SAS Language Reference: Dictionary** for descriptions and examples of additional character functions.

For example, in this DATA step, the value that results from the concatenation contains blanks because the length of the COLOR variable is eight:

```
data namegame;
      length color name $8 game $12;
      color='black';
      name='jack';
      game=color||name;
      put game=;
   run;
```

The value of GAME is **'black jack'** . To correct this problem, use the TRIM function in the concatenation operation as follows:

```
game=trim(color)||name;
```

This statement produces a value of **'blackjack'** for the variable GAME. The following additional examples demonstrate uses of the concatenation operator:

- If A has the value **'fortune'** , B has the value **'five'** , and C has the value **'hundred'** , then the following statement produces the value **'fortunefivehundred'** for the variable D:

  ```
  d=a||b||c;
  ```

- This example concatenates the value of a variable with a character constant.

  ```
  newname='Mr. or Ms. ' ||oldname;
  ```

  If the value of OLDNAME is **'Jones'** , then NEWNAME will have the value **'Mr. or Ms. Jones'** .

- Because the concatenation operation does not trim blanks, the following expression produces the value **'JOHN SMITH'** :

  ```
  name='JOHN   '||'SMITH';
  ```

- This example uses the PUT function to convert a numeric value to a character value. The TRIM function is used to trim blanks.

  ```
  month='sep   ';
  year=99;
  date=trim(month) || left(put(year,8.));
  ```

  The value of DATE is the character value **'sep99'** .

Order of Evaluation in Compound Expressions

Order of Evaluation in Compound Expressions shows the order of evaluation in compound expressions. The table contains the following columns:

Priority
: lists the priority of evaluation. In compound expressions, SAS evaluates the part of the expression containing operators in Group I first, then each group in order.

Order of Evaluation
: lists the rules governing which part of the expression SAS evaluates first. Parentheses are often used in compound expressions to group operands; expressions within parentheses are evaluated before those outside of them. The rules also list how a compound expression that contains more than one operator from the same group is evaluated.

Symbols
: lists the symbols that you use to request the comparisons, operations, and calculations.

Mnemonic Equivalent
: lists alternate forms of the symbol. In some cases, such as when your keyboard does not support special symbols, you should use the alternate form.

Definition
: defines the symbol.

Example
: provides an example of how to use the symbol or mnemonic equivalent in a SAS expression.

Order of Evaluation in Compound Expressions

Priority	Order of Evaluation	Symbols	Mnemonic Equivalent	Definition	Example
Group I	right to left	**		exponentiation (table note 1)	`y=a**2;`
		+		positive prefix (table note 2)	`y=+(a*b);`
		-		negative prefix (table note 3)	`z=-(a+b);`
		^ ¬ ~	NOT	logical not (table note 4)	`if not z then put x;`
		><	MIN	minimum (table note 5)	`x=(a><b);`
		<>	MAX	maximum	`x=(a<>b);`
Group II	left to right	*		multiplication	`c=a*b;`
		/		division	`f=g/h;`
Group III	left to right	+		addition	`c=a+b;`
		-		subtraction	`f=g-h;`
Group IV	left to right	|| ¦¦ !!		concatenate character values (table note 6)	`name= 'J'||'SMITH';`
Group V (table note 7)	left to right (table note 8)	<	LT	less than	`if x<y then c=5;`
		<=	LE	less than or equal to	`if x le y then a=0;`
		=	EQ	equal to	

					`if y eq (x+a) then output;`
		¬=	NE	not equal to	`if x ne z then output;`
		>=	GE	greater than or equal to	`if y>=a then output;`
		>	GT	greater than	`if z>a then output;`
			IN	equal to one of a list	`if state in ('NY','NJ','PA') then region='NE';` `y = x in (1:10);`
Group VI	left to right	&	AND	logical and	`if a=b & c=d` `then x=1;`
Group VII	left to right	| ¦ !	OR	logical or (table note 9)	`if y=2 or x=3 then a=d;`

TABLE NOTE 1: Because Group I operators are evaluated from right to left, the expression **x=2**3**4** is evaluated as **x=(2**(3**4))** .

TABLE NOTE 2: The plus (+) sign can be either a prefix or arithmetic operator. A plus sign is a prefix operator only when it appears at the beginning of an expression or when it is immediately preceded by an open parenthesis or another operator.

TABLE NOTE 3: The minus (-) sign can be either a prefix or arithmetic operator. A minus sign is a prefix operator only when it appears at the beginning of an expression or when it is immediately preceded by an open parenthesis or another operator.

TABLE NOTE 4: Depending on the characters available on your keyboard, the symbol can be the not sign (¬), tilde (~), or caret (^). The SAS system option CHARCODE allows various other substitutions for unavailable special characters.

TABLE NOTE 5: For example, the SAS System evaluates **-3><-3** as **-(3><-3)** , which is equal to **-(-3)** , which equals **+3** . This is because Group I operators are evaluated from right to left.

TABLE NOTE 6: Depending on the characters available on your keyboard, the symbol you use as the concatenation operator can be a double vertical bar (||), broken vertical bar (¦¦), or exclamation mark (!!).

TABLE NOTE 7: Group V operators are comparison operators. The result of a comparison operation is 1 if the comparison is true and 0 if it is false. Missing values are the lowest in any comparison operation.
The symbols =< (less than or equal to) are also allowed for compatibility with previous versions of the SAS System.
When making character comparisons, you can use a colon (:) after any of the comparison operators to compare only the first character or characters of the value. SAS truncates the longer value to the length of the shorter value during the comparison. For example, if **name=:'P'** compares the value of the first character of NAME to the letter P.

TABLE NOTE 8: An exception to this rule occurs when two comparison operators surround a quantity. For example, the expression **x<y<z** is evaluated as **(x<y)** and **(y<z)** .

TABLE NOTE 9: Depending on the characters available on your keyboard, the symbol you use for the logical or can be a single vertical bar (|), broken vertical bar (¦), or exclamation mark (!). You can also use the mnemonic equivalent OR.

Previous Page | Next Page | Top of Page