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 terminal.

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).

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 string by using a colon (:) after the comparison operator. 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';

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.

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 instance). 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.

IF Statement, Subsetting

Syntax

IF *expression*;

Arguments

expression is any SAS expression.

Details

The subsetting IF statement causes the DATA step to continue processing only those raw data records or those observations from a SAS data set that meet the condition of the expression that is specified in the IF statement. That is, if the expression is true for the observation or record (its value is neither 0 nor missing), SAS continues to execute statements in the DATA step and includes the current observation in the data set. The resulting SAS data set or data sets contain a subset of the original external file or SAS data set.

If the expression is false (its value is 0 or missing), no further statements are processed for that observation or record, the current observation is not written to the data set, and the remaining program statements in the DATA step are not executed. SAS immediately returns to the beginning of the DATA step because the subsetting IF statement does not require additional statements to stop processing observations.

Comparisons

- The subsetting IF statement is equivalent to this IF-THEN statement:
- ```
  if not (expression)
    then delete;
  ```

- When you create SAS data sets, use the subsetting IF statement when it is easier to specify a condition for including observations. When it is easier to specify a condition for excluding observations, use the DELETE statement.
- The subsetting IF and the WHERE statements are not equivalent. The two statements work differently and produce different output data sets in some cases. The most important differences are summarized as follows:
    - The subsetting IF statement selects observations that have been read into the program data vector. The WHERE statement selects observations before they are brought into the program data vector. The subsetting IF might be less efficient than the WHERE statement because it must read each observation from the input data set into the program data vector.
    - The subsetting IF statement and WHERE statement can produce different results in DATA steps that interleave, merge, or update SAS data sets.
    - When the subsetting IF statement is used with the MERGE statement, the SAS System selects observations after the current observations are combined. When
  ```

the WHERE statement is used with the MERGE statement, the SAS System applies the selection criteria to each input data set before combining the current observations.
- The subsetting IF statement can select observations from an existing SAS data set or from raw data that are read with the INPUT statement. The WHERE statement can select observations only from existing SAS data sets.
- The subsetting IF statement is executable; the WHERE statement is not.

Examples

- This example results in a data set that contains only those observations with the value F for the variable SEX:

```
if sex='F';
```

- This example results in a data set that contains all observations for which the value of the variable AGE is not missing or 0:

```
if age;
```

IF-THEN/ELSE Statement

Syntax

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

Arguments

expression is any SAS expression and is a required argument.

statement can be any executable SAS statement or DO group.

Details

SAS evaluates the expression in an IF-THEN statement to produce a result that is either non-zero, zero, or missing. A non-zero and nonmissing result causes the expression to be true; a result of zero or missing causes the expression to be false.

If the conditions that are specified in the IF clause are met, the IF-THEN statement executes a SAS statement for observations that are read from a SAS data set, for records in an external file, or for computed values. An optional ELSE statement gives an alternative action if the THEN clause is not executed. The ELSE statement, if used, must immediately follow the IF-THEN statement.

Using IF-THEN statements without the ELSE statement causes SAS to evaluate all IF-THEN statements. Using IF-THEN statements with the ELSE statement causes SAS to execute IF-THEN statements until it encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

Note: For greater efficiency, construct your IF-THEN/ELSE statement with conditions of decreasing probability

Comparisons

- Use a SELECT group rather than a series of IF-THEN statements when you have a long series of mutually exclusive conditions.
- Use subsetting IF statements, without a THEN clause, to continue processing only those observations or records that meet the condition that is specified in the IF clause.

Examples

These examples show different ways of specifying the IF-THEN/ELSE statement.

- ```
  if x then delete;
  ```

- ```
  if status='OK' and type=3 then count+1;
  ```

- ```
  if age ne agecheck then delete;
  ```

- ```
  if x=0 then
     if y ne 0 then put 'X ZERO, Y NONZERO';
     else put 'X ZERO, Y ZERO';
  else put 'X NONZERO';
  ```

- ```
  if answer=9 then
     do;
        answer=.;
        put 'INVALID ANSWER FOR ' id=;
     end;
  else
     do;
        answer=answer10;
        valid+1;
     end;
  ```

- ```
  data region;
     input city $ 1-30;
     if city='New York City'
        or city='Miami' then
        region='ATLANTIC COAST';
     else if city='San Francisco'
        or city='Los Angeles' then
           region='PACIFIC COAST';
     datalines;
  ...more data lines...
  ;
  ```