CHAPTER 12

Expressions

Definitions

expression
is generally a sequence of operands and operators that form a set of instructions that are performed to produce a resulting value. You use expressions in SAS program statements to create variables, assign values, calculate new values,

transform variables, and perform conditional processing. SAS expressions can resolve to numeric values, character values, or Boolean values.

operands
are constants or variables that can be numeric or character.

operators
are symbols that represent a comparison, arithmetic calculation, or logical operation; a SAS function; or grouping parentheses.

simple expression
is an expression with no more than one operator. A simple expression can consist of a single

- constant
- variable
- function.

compound expression
is an expression that includes several operators. When SAS encounters a compound expression, it follows rules to determine the order in which to evaluate each part of the expression.

The following are examples of SAS expressions:

- **3**
- **x**
- **x+1**
- **age<100**
- **trim(last)||', '||first**

SAS Constants in Expressions

Definition

A SAS *constant* is a number or a character string that indicates a fixed value. Constants can be used as expressions in many SAS statements, including variable assignment and IF-THEN statements. They can also be used as values for certain options. Constants are also called *literals*.

The following are types of SAS constants:

- character
- numeric
- date, time, and datetime
- bit testing.

Character Constants

A *character constant* consists of 1 to 32,767 characters and must be enclosed in quotation marks. Character constants can also be represented in hexadecimal form.

Using Quotation Marks

In the following SAS statement, **Tom** is a character constant:

```
if name='Tom' then do;
```

If a character constant includes a single quotation mark, surround it with double quotation marks. For example, to specify the character value **`Tom's`** as a constant, enter

```
name="Tom's"
```

You can also write a single quotation mark as two consecutive single quotation marks and SAS treats it as one. You can then surround the character constant with single quotation marks:

```
name='Tom''s'
```

The same principle holds true for double quotation marks:

```
name="Tom""s"
```

CAUTION:

Matching quotation marks correctly is important. Missing or extraneous quotation marks cause SAS to misread both the erroneous statement and the statements that follow it. For example, in **`name='O'Brien';`**, **`O`** is the character value of NAME, **`Brien`** is extraneous, and **`';`** begins another quoted string. △

Comparing Character Constants and Character Variables

It is important to remember that character constants are enclosed in quotation marks, but names of character variables are not. This distinction applies wherever you can use a character constant, such as in titles, footnotes, labels, and other descriptive strings; in option values; and in operating environment-specific strings, such as file specifications and commands.

The following statements use character constants:

- **`x='abc';`**
- **`if name='Smith' then do;`**

The following statements use character variables:

- **`x=abc;`**
- **`if name=Smith then do;`**

In the second set of examples, SAS searches for variables named ABC and SMITH, instead of constants.

Note: SAS distinguishes between uppercase and lowercase when comparing quoted values. For example, the character values **`'Smith'`** and **`'SMITH'`** are not equivalent. △

Hexadecimal Notation

SAS character constants can be expressed in hexadecimal notation. A character hex constant is a string of an even number of hex characters enclosed in single or double quotation marks, followed immediately by an X, as in this example:

```
'534153'x
```

A comma can be used to make the string more readable, but it is not part of and does not alter the hex value. If the string contains a comma, the comma must separate an even number of hex characters within the string, as in this example:

```
if value='3132,3334'x then do;
```

Numeric Constants

A *numeric constant* is a number that appears in a SAS statement. Numeric constants can be presented in many forms, including

- standard notation
- scientific (E) notation
- hexadecimal notation.

Standard Notation

Most numeric constants are written just as numeric data values are. The numeric constant in the following expression is 100:

```
part/all*100
```

Numeric constants expressed in standard notation can be integers, can be specified with or without a plus or minus sign, and can include decimal places, as in these examples:

- **1**
- **1.23**
- **01**
- **-5**

Scientific Notation

In scientific notation, the number before the E is multiplied by the power of ten that is indicated by the number after the E. For example, 2E4 is the same as $2x10^4$ or 20,000. For numeric constants larger than $(10^{32})–1$, you must use scientific notation. Additional examples follow:

- **1.2e23**
- **0.5e-10**

Hexadecimal Notation

A numeric constant that is expressed as a hexadecimal value starts with a numeric digit (usually 0), can be followed by more hexadecimal digits, and ends with the letter X. The constant can contain up to 16 valid hexadecimal digits (0 to 9, A to F). The following are numeric hex constants:

- **0c1x**
- **9x**

You can use numeric hex constants in a DATA step, as follows:

```
data test;
   input abend pib2.;
   if abend=0c1x or abend=0b0ax then do;
    … more SAS statements …
 run;
```

Date, Time, and Datetime Constants

You can create a *date constant*, *time constant*, or *datetime constant* by specifying the date or time in single or double quotation marks, followed by a D (date), T (time), or DT (datetime) to indicate the type of value. Use the following patterns to create date and time constants:

'*ddmmm<yy>yy*'D or "*ddmmm<yy>yy*"D represents a SAS date value:

- **`date='1jan2006'd;`**
- **`date='01jan04'd;`**

'*hh:mm<:ss.s>*'T or "*hh:mm<:ss.s>*"T represents a SAS time value:

- **`time='9:25't;`**
- **`time='9:25:19pm't;`**

'*ddmmm<yy>yy*:*hh:mm<:ss.s>*'DT or "*ddmmm<yy>yy*:*hh:mm<:ss.s>*"DT represents a SAS datetime value:

- **`if begin='01may04:9:30:00'dt then end='31dec90:5:00:00'dt;`**
- **`dtime='18jan2002:9:27:05am'dt;`**

For more information on SAS dates, refer to Chapter 13, "Dates, Times, and Intervals," on page 147.

Bit Testing Constants

Bit masks are used in bit testing to compare internal bits in a value's representation. You can perform bit testing on both character and numeric variables. The general form of the operation is:

expression comparison-operator bit-mask

The following are the components of the bit-testing operation:

expression
: can be any valid SAS expression. Both character and numeric variables can be bit tested. When SAS tests a character value, it aligns the left-most bit of the mask with the left-most bit of the string; the test proceeds through the corresponding bits, moving to the right. When SAS tests a numeric value, the value is truncated from a floating-point number to a 32-bit integer. The right-most bit of the mask is aligned with the right-most bit of the number, and the test proceeds through the corresponding bits, moving to the left.

comparison-operator
: compares an expression with the bit mask. Refer to "Comparison Operators" on page 138 for a discussion of these operators.

bit-mask
: is a string of 0s, 1s, and periods in quotation marks that is immediately followed by a B. Zeros test whether the bit is off; ones test whether the bit is on; and periods ignore the bit. Commas and blanks can be inserted in the bit mask for readability without affecting its meaning.

CAUTION:
Truncation can occur when SAS uses a bit mask. If the expression is longer than the bit mask, SAS truncates the expression before it compares it with the bit mask. A false comparison may result. An expression's length (in bits) must be less than or equal to the length of the bit mask. If the bit mask is longer than a character expression, SAS prints a warning in the log, stating that the bit mask is truncated on the left, and continues processing. △

The following example tests a character variable:

```
if a='..1.0000'b then do;
```

If the third bit of A (counting from the left) is on, and the fifth through eighth bits are off, the comparison is true and the expression result is 1. Otherwise, the comparison is false and the expression result is 0. The following is a more detailed example:

```
data test;
  input @88 bits $char1.;
  if bits='10000000'b
    then category='a';
  else if bits='01000000'b
    then category='b';
  else if bits='00100000'b
    then category='c';
   run;
```

Note: Bit masks cannot be used as bit literals in assignment statements. For example, the following statement is not valid:

```
x='0101'b;     /* incorrect */
```

△

The $BINARY*w*. and BINARY*w*. formats and the $BINARY*w*., BINARY*w.d*, and BITS*w.d* informats can be useful for bit testing. You can use them to convert character and numeric values to their binary values, and vice versa, and to extract specified bits from input data. See *SAS Language Reference: Dictionary* for complete descriptions of these formats and informats.

SAS Variables in Expressions

Definition

variable
is a set of data values that describe a given characteristic. A variable can be used in an expression.

Automatic Numeric-Character Conversion

If you specify a variable in an expression, but the variable value does not match the type called for, SAS attempts to convert the value to the expected type. SAS automatically converts character variables to numeric variables and numeric variables to character variables, according to the following rules:

- If you use a character variable with an operator that requires numeric operands, such as the plus sign, SAS converts the character variable to numeric.
- If you use a comparison operator, such as the equal sign, to compare a character variable and a numeric variable, the character variable is converted to numeric.
- If you use a numeric variable with an operator that requires a character value, such as the concatenation operator, the numeric value is converted to character using the BEST12. format. Because SAS stores the results of the conversion beginning with the right-most byte, you must store the converted values in a

variable of sufficient length to accommodate the BEST12. format. You can use the LEFT function to left-justify a result.

- If you use a numeric variable on the left side of an assignment statement and a character variable on the right, the character variable is converted to numeric. In the opposite situation, where the character variable is on the left and the numeric is on the right, SAS converts the numeric variable to character using the BEST*n*. format, where *n* is the length of the variable on the left.

When SAS performs an automatic conversion, it prints a note in the SAS log informing you that the conversion took place. If converting a character variable to numeric produces invalid numeric values, SAS assigns a missing value to the result, prints an error message in the log, and sets the value of the automatic variable _ERROR_ to 1.

Note: You can also use the PUT and INPUT functions to convert data values. These functions can be more efficient than automatic conversion. See "The Concatenation Operator" on page 143 for an example of the PUT function. See *SAS Language Reference: Dictionary* for more details on these functions. △

For more information on SAS variables, see Chapter 10, "SAS Variables," on page 99 or the *SAS Language Reference: Dictionary*.

SAS Functions in Expressions

A SAS *function* is a keyword that you use to perform a specific computation or system manipulation. Functions return a value, might require one or more arguments, and can be used in expressions. For further information on SAS functions, see *SAS Language Reference: Dictionary*.

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 kinds 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 Chapter 18, "WHERE-Expression Processing," on page 229.

Arithmetic Operators

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

Table 12.1 Arithmetic Operators

Symbol	Definition	Example	Result
**	exponentiation	**a**3**	raise A to the third power
*	multiplication[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

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 Chapter 11, "Missing Values," on page 123 for a discussion of how to prevent the propagation of missing values.

See "Order of Evaluation in Compound Expressions" on page 144 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 comparision is true, the result is 1. If the comparision 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:

Table 12.2 Comparison Operators

Symbol	Mnemonic Equivalent	Definition	Example
=	EQ	equal to	**a=3**
^=	NE	not equal to[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[2]	**sales>=300**
<=	LE	less than or equal to[3]	**sales<=100**
	IN	equal to one of a list	**num in (3, 4, 5)**

1 The symbol you use for NE depends on your terminal.
2 The symbol => is also accepted for compatibility with previous releases of SAS.
3 The symbol =< is also accepted for compatibility with previous releases of SAS.

See "Order of Evaluation in Compound Expressions" on page 144 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" on page 140 for details. △

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 Chapter 10, "SAS Variables," on page 99 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 Chapter 11, “Missing Values,” on page 123 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';`**

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;`**

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:

Table 12.3 Logical Operators

Symbol	Mnemonic Equivalent	Example
&	AND	**`(a>b & c>d)`**
|	OR[1]	**`(a>b or c>d)`**
!	OR	
¦	OR	
¬	NOT[2]	**`not(a>b)`**
ˆ	NOT	
~	NOT	

1 The symbol you use for OR depends on your operating environment.
2 The symbol you use for NOT depends on your operating environment.

See “Order of Evaluation in Compound Expressions” on page 144 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” on page 142.

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” on page 142). 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 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 described in "Order of Missing Values" on page 125. For example, the maximum value returned by .A<>.Z is the value .Z.

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

Table 12.4 on page 144 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.

Table 12.4 Order of Evaluation in Compound Expressions

Priority	Order of Evaluation	Symbols	Mnemonic Equivalent	Definition	Example
Group I	right to left	**		exponentiation[1]	**y=a**2;**
		+		positive prefix[2]	**y=+(a*b);**
		-		negative prefix[3]	**z=-(a+b);**
		ˆ ¬ ˜	NOT	logical not[4]	**if not z then put x;**
		><	MIN	minimum[5]	**x=(a><b);**
		<>	MAX	maximum	**x=(a<>b);**

Priority	Order of Evaluation	Symbols	Mnemonic Equivalent	Definition	Example
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[6]	`name= 'J'‖'SMITH';`
Group V[7]	left to right[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';`
Group VI	left to right	&	AND	logical and	`if a=b & c=d then x=1;`
Group VII	left to right	| ¦ !	OR	logical or[9]	`if y=2 or x=3 then a=d;`

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

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 a left parenthesis or another operator.

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 a left parenthesis or another operator.

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.

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.

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 (!!).

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(s) 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.

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)`**.

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.

WHERE Expressions

A *WHERE expression* is a type of SAS expression that is used within a WHERE statement or WHERE= data set option to specify a condition for selecting observations for processing in a DATA or PROC step. For syntax and further information on WHERE expressions, see Chapter 18, “WHERE-Expression Processing,” on page 229 and *SAS Language Reference: Dictionary*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

Copyright © 1999 SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–441–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, November 1999