CHAPTER

6

Macro Expressions

Introduction

There are three types of macro expressions: text, logical, and arithmetic. A *text expression* is any combination of text, macro variables, macro functions, or macro calls. Text expressions are resolved to generate text. Here are some examples of text expressions:

- &BEGIN
- %GETLINE
- &PREFIX.PART&SUFFIX
- %UPCASE(&ANSWER)

A *logical expression* or an *arithmetic expression* is a sequence of operators and operands forming a set of instructions that are evaluated to produce a result. An arithmetic expression contains an arithmetic operator. A logical expression contains a logical operator. The following table show examples of simple arithmetic and logical expressions:

Arithmetic Expressions	Logical expressions
1 + 2	&DAY = FRIDAY
4 * 3	A < a
4 / 2	1 < &INDEX
00FFx - 003Ax	&START NE &END

The following sections tell you where and how to use arithmetic and logical expressions in macro functions and statements.

Arithmetic and Logical Expressions

You can use arithmetic and logical expressions in specific macro functions and statements (Table 6.1 on page 64). The arithmetic and logical expressions in these functions and statements enable you to control the text generated by a macro when it is executed.

Table 6.1 Macro Language Elements that Evaluate Arithmetic and Logical Expressions

%DO*macro-variable=expression* %TO *expression*<%BY *expression*>;
%DO %UNTIL(*expression*);
%DO %WHILE(*expression*);
%EVAL (*expression*);
%IF *expression* %THEN *statement*;
%QSCAN(*argument,expression*<,*delimiters*>)
%QSUBSTR(*argument,expression*<,expression>)
%SCAN(*argument,expression*,<delimiters>)
%SUBSTR(*argument,expression*<,expression>)
%SYSEVALF(*expression,conversion-type*)

You can use text expressions to generate partial or complete arithmetic or logical expressions. The macro processor resolves text expressions before it evaluates the arithmetic or logical expressions. For example, when you submit the following statements, the macro processor resolves the macro variables &A, &B, and &OPERATOR in the %EVAL function, before it evaluates the expression 2 + 5:

```
%let A=2;
%let B=5;
%let operator=+;
%put The result of &A &operator &B is %eval(&A &operator &B).;
```

When you submit these statements, the %PUT statement writes this line to the log:

```
The result of 2 + 5 is 7.
```

Operands and Operators

Operands in arithmetic or logical expressions are always text. However, an operand that represents a number can be temporarily converted to a numeric value when an expression is evaluated. By default, the macro processor uses integer arithmetic, and only integers and hexadecimal values that represent integers can be converted to a numeric value. Operands that contain a period character, for example 1.0, are not converted. The exception is the %SYSEVALF function, which interprets a period character in its argument as a decimal point and converts the operand to a floating point value on your operating system.

Operators in macro expressions are a subset of those in the DATA step (Table 6.2 on page 65). However, in macro, there is no MAX or MIN operator, and macro does not

recognize IN or ':', as does the DATA step. The order in which operations are performed when an expression is evaluated is the same in the macro language as in the DATA step. Operations within parentheses are performed first.

Note: Expressions in which comparison operators surround a macro expression, as in 10<&X<20, may or may not be the equivalent of a DATA step compound expression (depending on what the expression resolves into). To be safe, write the connecting operator explicitly, as in the expression 10<&X AND &X<20. △

Table 6.2 Macro Language Operators

Operator	Mnemonic	Name	Procedence
**		exponential	1
+		positive prefix	2
—		negative prefix	2
¬^~	NOT	logical not*	3
*		multiplication	4
/		division	4
+		addition	5
—		subtraction	5
<	LT	less than	6
<=	LE	less than or equal	6
=	EQ	eqaul	6
¬=^=~=	NE	not equal*	6
>	GT	greater than	6
>=	GE	greater than or equal	6
&	AND	logical and	7
|	OR	logical or	8

*The symbol to use depends on your keyboard.

How the Macro Processor Evaluates Arithmetic Expressions

The macro facility is a string handling facility. However, in specific situations, the macro processor can evaluate operands that represent numbers as numeric values. When the macro processor evaluates an expression that contains an arithmetic operator and operands that represent numbers, it temporarily converts the operands to numeric values and performs the integer arithmetic operation. The result of the evaluation is text.

Evaluating Numeric Operands

By default, arithmetic evaluation in most macro statements and functions is performed with integer arithmetic. The exception is the %SYSEVALF function. See

"Evaluating Floating Point Operands" on page 66 for more information. The following macro statements illustrate integer arithmetic evaluation:

```
%let a=%eval(1+2);
%let b=%eval(10*3);
%let c=%eval(4/2);
%let i=%eval(5/3);
%put The value of a is &a;
%put The value of b is &b;
%put The value of c is &c;
%put The value of I is &i;
```

When you submit these statements, the following messages appear in the log:

```
The value of a is 3
The value of b is 30
The value of c is 2
The value of I is 1
```

Notice the result of the last statement. If you perform division on integers that would ordinarily result in a fraction, integer arithmetic discards the fractional part.

When the macro processor evaluates an integer arithmetic expression that contains a character operand, it generates an error. Only operands that contain characters that represent integers or hexadecimal values are converted to numeric values. The following statement shows an incorrect usage:

```
%let d=%eval(10.0+20.0);    /*INCORRECT*/
```

Because the %EVAL function supports only integer arithmetic, the macro processor does not convert a value containing a period character to a number, and the operands are evaluated as character operands. This statement produces the following error message:

```
ERROR: A character operand was found in the %EVAL function or %IF
condition where a numeric operand is required. The condition was:
10.0+20.0
```

Evaluating Floating Point Operands

The %SYSEVALF function evaluates arithmetic expressions with operands that represent floating point values. For example, the following expressions in the %SYSEVALF function are evaluated using floating point arithmetic:

```
%let a=%sysevalf(10.0*3.0);
%let b=%sysevalf(10.5+20.8);
%let c=%sysevalf(5/3);
%put 10.0*3.0 = &a;
%put 10.5+20.8 = &b;
%put 5/3 = &c;
```

The %PUT statements display the following messages in the log:

```
10.0*3.0 = 30
10.5+20.8 = 31.3
5/3 = 1.6666666667
```

When the %SYSEVALF function evaluates arithmetic expressions, it temporarily converts the operands that represent numbers to floating point values. The result of the

evaluation can represent a floating point value, but as in integer arithmetic expressions, the result is always text.

The %SYSEVALF function provides conversion type specifications: BOOLEAN, INTEGER, CEIL, and FLOOR. For example, these %PUT statements

```
%let a=2.5;
%put %sysevalf(&a,boolean);
%put %sysevalf(&a,integer);
%put %sysevalf(&a,ceil);
%put %sysevalf(&a,floor);
```

return 1, 2, 3, and 2, respectively. These conversion types tailor the value returned by %SYSEVALF so that it can be used in other macro expressions that require integer or Boolean values.

CAUTION:

Specify a conversion type for the %SYSEVALF function. If you use the %SYSEVALF function in macro expressions or assign its results to macro variables that are used in other macro expressions, then errors or unexpected results may occur if the %SYSEVALF function returns missing or floating point values. To prevent errors, specify a conversion type that returns a value compatible with other macro expressions. See "%SYSEVALF" in Chapter 13, "Macro Language Dictionary" for more information on using conversion types. △

How the Macro Processor Evaluates Logical Expressions

A logical, or Boolean, expression returns a value that is evaluated as true or false. In the macro language, any numeric value other than 0 is true and a value of 0 is false.

Comparing Numeric Operands in Logical Expressions

When the macro processor evaluates logical expressions that contain operands that represent numbers, it converts the characters temporarily to numeric values. To illustrate how the macro processor evaluates logical expressions with numeric operands, consider the following macro definition:

```
%macro compnum(first,second);
   %if &first>&second %then %put &first is greater than &second;
   %else %if &first=&second %then %put &first equals &second;
   %else %put &first is less than &second;
%mend compnum;
```

Invoking the COMPNUM macro with these values

```
%compnum(1,2)
%compnum(-1,0)
```

displays these results in the log:

```
1 is less than 2
-1 is less than 0
```

The results show that the operands in the logical expressions were evaluated as numeric values.

Comparing Floating Point or Missing Values

You must use the %SYSEVALF function to evaluate logical expressions containing floating point or missing values. To illustrate comparisons with floating point and missing values, consider the following macro that compares parameters passed to it with the %SYSEVALF function and places the result in the log:

```
%macro compflt(first,second);
   %if %sysevalf(&first>&second) %then %put &first is greater than &second;
   %else %if  %sysevalf(&first=&second) %then %put &first equals &second;
   %else %put %sysevalf(&first is less than &second);
%mend compflt;
```

Invoking the COMPFLT macro with these values

```
%compflt (1.2,.9)
%compflt (-.1,.)
%compflt (0,.)
```

places these values in the log:

```
1.2 is greater than .9
-.1 is greater than .
0 is greater than .
```

The results show that the %SYSEVALF function evaluated the floating point and missing values.

Comparing Character Operands in Logical Expressions

To illustrate how the macro processor evaluates logical expressions, consider the COMPC macro. Invoking the COMPC macro compares the values passed as parameters and places the result in the log.

```
%macro compchar(first,second);
   %if &first>&second %then %put &first comes after &second;
   %else %put &first comes before &second;
%mend compchar;
```

Invoking the macro COMPCHAR with these values

```
%compchar(a,b)
%compchar(.,1)
%compchar(Z,E)
```

prints these results in the log:

```
a comes before b
. comes before 1
Z comes after E
```

When the macro processor evaluates expressions with character operands, it uses the sort sequence of the host operating system for the comparison. The comparisons in these examples work with both EBCDIC and ASCII sort sequences.

A special case of a character operand is an operand that looks numeric but contains a period character. If you use an operand with a period character in an expression, both operands are compared as character values. This can lead to unexpected results. So that you can understand and better anticipate results, look at the following examples.

If you invoke the COMPNUM macro, shown earlier, with these values

```
%compnum(10,2.0)
```

then these values are written to the log:

```
10 is less than 2.0
```

Because the %IF-THEN statement in the COMPNUM macro uses integer evaluation, it does not convert the operands with decimal points to numeric values. The operands are compared as character strings using the host sort sequence, which is the comparison of characters with smallest-to-largest values. For example, lowercase letters may have smaller values than uppercase, and uppercase letters may have smaller values than digits.

CAUTION:

The host sort sequence determines comparison results. If you use a macro definition on more than one operating system, comparison results may differ because the sort sequence of one host operating system may differ from the other system. Refer to "The SORT Procedure" in *SAS Procedures Guide* for more information on host sort sequences. △

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Macro Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 310 pages.

SAS Macro Language: Reference, Version 8

1-58025-522-1

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

1st printing, October 1999