Chapter 3

Expressions

This chapter contains the following subsections:

- "Arithmetic Expressions"
- "Character Expressions"
- "Relational Expressions"
- "Logical Expressions"
- "Evaluating Expressions in General"

An expression performs a specified type of computation. It is composed of a sequence of operands, operators, and parentheses. The types of Fortran expressions are

- arithmetic
- character
- relational
- logical

This chapter describes formation, interpretation, and evaluation rules for each type of expression. This chapter also discusses mixed−mode expressions, which are Fortran 77 enhancements of Fortran 66.

Arithmetic Expressions

An arithmetic expression specifies a numeric computation that yields a numeric value on evaluation. The simplest form of an arithmetic expression can be:

- an unsigned arithmetic constant
- a symbolic name of an arithmetic constant
- an arithmetic variable reference
- an arithmetic array element reference
- an arithmetic function reference

You can form more complicated arithmetic expressions from one or more operands together with arithmetic operators and parentheses.

An arithmetic element can include logical entities because logical data is treated as integer data when used in an arithmetic context. When both arithmetic and logical operands exist for a given operator, the logical operand is promoted to type **INTEGER** of the same byte length as the original logical length. For example, a **LOGICAL*2** will be promoted to **INTEGER*2** and a **LOGICAL*4** will be promoted to **INTEGER*4**.

Arithmetic Operators

Table 3−1shows the arithmetic operators.

Table 3−1 Arithmetic Operators

Operator	Function
**	Exponentiation
*	Multiplication
/	Division
+	Addition or identity
-	Subtraction or negation

Use the exponentiation, division, and multiplication operators between exactly two operands. You can use the addition and subtraction operators with one or two operands; in the latter case, specify the operator before the operand; for example, **-TOTAL**.

Do not specify two operators in succession. (Note that the exponentiation operator consists of the two characters (**), but is a *single* operator.) Implied operators, as in implied multiplication, are not allowed.

Interpretation of Arithmetic Expressions

Table 3−2interprets sample arithmetic expressions.

Table 3−2 Interpretation of Arithmetic Expressions

Operator	Use	Interpretation
**	x1 ** x2	Exponentiate x1 to the power of x2
*	x1 * x2	Multiply x1 and x2
/	x1 / x2	Divide x1 by x2
+	x1 + x2	Add x1 and x2
	+ x	x (identity)
-	x1 - x2	Subtract x1 from x2
	−x	Negate x

An arithmetic expression containing two or more operators is interpreted based on a precedence relation among the arithmetic operators. This precedence, from highest to lowest, is

- ( )
- **
- * and /
- + and -

Use parentheses to override the order of precedence.

The following is an example of an arithmetic expression:

```
A/B−C**D
```

The operators are executed in the following sequence:

1. **C**D** is evaluated first.
2. **A/B** is evaluated next.
3. The result of **C**D** is subtracted from the result of **A/B** to give the final result.

A unary operator (-) can follow another operator. Specifying the unary operator after the exponentiation operator produces a variation on the standard order of operations. The unary operator is evaluated first in that case, resulting in exponentiation taking a lower precedence in the expression.

For example, the following expression

```
A ** – B * C
```

is interpreted as

```
A ** ( – B * C )
```

Arithmetic Operands

Arithmetic operands must specify values with integer, real, double–precision, complex, or double–complex data types. You can combine specific operands in an arithmetic expression. The arithmetic operands, in order of increasing complexity, are

- primary
- factor
- term
- arithmetic expression

A *primary* is the basic component in an arithmetic expression. The forms of a primary are

- an unsigned arithmetic constant
- a symbolic name of an arithmetic constant
- an arithmetic variable reference
- an arithmetic array element reference
- an arithmetic function reference
- an arithmetic expression enclosed in parentheses

A *factor* consists of one or more primaries separated by the exponentiation operator. The forms of a factor are

- primary
- primary ** factor

Factors with more than one exponentiation operator are interpreted from right to left. For example, **I**J**K** is interpreted as **I**(J**K)**, and **I**J**K**L** is interpreted as **I**(J**(K**L))**.

The *term* incorporates the multiplicative operators into arithmetic expressions. Its forms are

- factor
- term/factor
- term * factor

The above definition indicates that factors are combined from left to right in a term containing two or more multiplication or division operators.

Finally, at the highest level of the hierarchy, are the *arithmetic expressions*. The forms of an arithmetic expression are

- term
- + term
- - term
- arithmetic expression + term
- arithmetic expression - term

An arithmetic expression consists of one or more terms separated by an addition operator or a subtraction operator. The terms are combined from left to right. For example, **A+B−C**has the same interpretation as the expression **(A+B)−C** Expressions such as **A*−B**and **A+−B**are not allowed. The correct forms are **A*(-B)** and **A+(−B)**

An arithmetic expression can begin with a plus or minus sign.

Arithmetic Constant Expressions

An *arithmetic constant expression* is an arithmetic expression containing no variables. Therefore, each primary in an arithmetic constant expression must be one of the following:

- arithmetic constant
- symbolic name of an arithmetic constant
- arithmetic constant expression enclosed in parentheses

In an arithmetic constant expression, do not specify the exponentiation operator unless the exponent is of type integer. Variable, array element, and function references are not allowed. Examples of integer constant expressions are

```
 7
-7
-7+5
3**2
x+3 (where x is the symbolic name of a constant)
```

Integer Constant Expressions

An *integer constant expression* is an arithmetic constant expression containing only integers. It can contain constants or symbolic names of constants, provided they are of type integer. As with all constant expressions, no variables, array elements, or function references are allowed.

Evaluating Arithmetic Expressions

The data type of an expression is determined by the data types of the operands and functions that are referenced. Thus, integer expressions, real expressions, double−precision expressions, complex expressions, and double expressions have values of type integer, real, double−precision, complex, and double−complex, respectively.

Single−Mode Expressions

Single−mode expressions are arithmetic expressions in which all operands have the same data type. The data type of the value of a single−mode expression is thus the same as the data type of the operands. When the addition operator or the subtraction operator is used with a single operand, the data type of the resulting expression is the same as the data type of the operand.

Mixed−Mode Expressions

Mixed−mode expressions contain operands with two or more data types. The data type of the result of a mixed−mode expression depends on the rank associated with each data type, as shown inTable 3−3

Table 3−3 Data Type Ranks

Data Type	Rank
INTEGER*1	1 (lowest)
INTEGER*2	2
INTEGER*4	3
REAL*4	4
REAL*8 (double precision)	5
COMPLEX*8	6
COMPLEX*16	7 (highest)

Except for exponentiation (discussed below), the result of a mixed−mode expression is assigned the data type of the highest−ranked element in the expression. The lower−ranked operand is converted to the type of the higher−ranked operand so that the operation is performed on values with equivalent data types. For example, an operation on an integer operand and a real operand produces a result of type real.

Operations that combine **REAL*8** (**DOUBLE PRECISION**) and **COMPLEX*8** (**COMPLEX**) are not allowed. The **REAL*8** operand must be explicitly converted (for example, by using the **SNGL** intrinsic function).

Exponentiation

Exponentiation is an exception to the above rules for mixed−mode expressions. When raising a value to an integer power, the integer is not converted. The result is assigned the type of the left operand.

When a complex value is raised to a complex power, the value of the expression is defined as follows:

```
xy = EXP (y * LOG(x))
```

Integer Division

One operand of type integer can be divided by another operand of type integer. The result of an integer division operation is a value of type integer, referred to as an integer quotient. The integer quotient is obtained as follows:

- If the magnitude of the mathematical quotient is less than one, then the integer quotient is zero. For example, the value of the expression (18/30) is zero.
- If the magnitude of the mathematical quotient is greater than or equal to one, then the integer quotient is the largest integer that does not exceed the magnitude of the mathematical quotient and whose sign is the same as that of the mathematical quotient. For example, the value of the expression (-9/2) is -4.

Character Expressions

A *character expression* yields a character string value on evaluation. The simplest form of a character expression can be one of these types of characters:

- constant
- variable reference
- array element reference
- substring reference
- function reference

Construct complicated character expressions from one or more operands together with the concatenate operator and parentheses.

Concatenate Operator

The concatenate operator (//) is the only character operator defined in Fortran. A character expression formed from the concatenation of two character operands *x1* and *x2* is specified as

x1 // x2

The result of this operation is a character string with a value of *x1* extended on the right with the value of *x2*. The length of the result is the sum of the lengths of the character operands. For example,

```
'HEL' // 'LO2'
```

The result of the above expression is the string **HELLO2** of length six.

Character Operands

A *character operand* must identify a value of type character and must be a character expression. The basic component in a character expression is the character *primary*. The forms of a character primary are

- character constant
- symbolic name of a character constant
- character variable reference
- character array element reference
- character substring reference
- character function reference
- character expression enclosed in parentheses

A *character expression* consists of one or more character primaries separated by the concatenation operator. Its forms are

- character primary
- character expression // character primary

In a character expression containing two or more concatenation operators, the primaries are combined from left to right. Thus, the character expression

```
'A' // 'BCD' // 'EF'
```

is interpreted the same as

```
('A' // 'BCD') // 'EF'
```

The value of the above character expression is **ABCDEF**.

Except in a character assignment statement, concatenation of an operand with an asterisk (*) as its length specification is not allowed unless the operand is the symbolic name of a constant.

Character Constant Expressions

A *character constant* expression is made up of operands that cannot vary. Each primary in a character constant expression must be a

- character constant
- symbolic name of a character constant
- character constant expression enclosed in parentheses

A character constant expression cannot contain variable, array element, substring, or function references.

Relational Expressions

A relational expression yields a logical value of either **.TRUE.** or **.FALSE.** on evaluation and comparison of two arithmetic expressions or two character expressions. A relational expression can appear only within a logical expression. Refer to "Logical Expressions" for details about logical expressions.

Relational Operators

Table 3–4lists the Fortran relational operators.

Table 3–4 Fortran Relational Operators

Relational Operator	Meaning
.EQ.	Equal to
.NE.	Not equal to
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to

Arithmetic and character operators are evaluated *before* relational operators.

Relational Operands

The operands of a relational operator can be arithmetic or character expressions. The relational expression requires exactly two operands and is written in the following form:

e1 relop e2

where

e1 and *e2* are arithmetic or character expressions.

relop is the relational operator.

Note: Both *e1* and *e2* must be the same type of expression, either arithmetic or character.

Evaluating Relational Expressions

The result of a relational expression is of type logical, with a value of **.TRUE.** or **.FALSE.**. The manner in which the expression is evaluated depends on the data type of the operands.

Arithmetic Relational Expressions

In an arithmetic relational expression, *e1* and *e2* must each be an integer, real, double precision, complex, or double complex expression. *relop* must be a relational operator.

The following are examples of arithmetic relational expressions:

```
(a + b) .EQ. (c + 1)
HOURS .LE. 40
```

You can use complex type operands only when specifying either the **.EQ**. or **.NE.** relational operator.

An arithmetic relational expression has the logical value **.TRUE.** only if the values of the operands satisfy the relation specified by the operator. Otherwise, the value is **.FALSE.**.

If the two arithmetic expressions *e1* and *e2* differ in type, the expression is evaluated as follows:

```
((e1) - (e2)) relop 0
```

where the value 0 (zero) is of the same type as the expression ((*e1*)– (*e2*)) and the type conversion rules apply to the expression. Do not compare a double precision value with a complex value.

Character Relational Expressions

In a character relational expression, *e1* and *e2* are character expressions and *relop* is a relational operator.

The following is an example of a character relational expression:

```
NAME .EQ. 'HOMER'
```

A character relational expression has the logical value **.TRUE.** only if the values of the operands satisfy the relation specified by the operator. Otherwise, the value is **.FALSE.**. The result of a character relational expression depends on the collating sequence as follows:

- If *e1* and *e2* are single characters, their relationship in the collating sequence determines the value of the operator. *e1* is less than or greater than *e2* if *e1* is before or after *e2*, respectively, in the collating sequence.
- If either *e1* or *e2* are character strings with lengths greater than 1, corresponding individual characters are compared from left to right until a relationship other than **.EQ.** can be determined.
- If the operands are of unequal length, the shorter operand is extended on the right with blanks to the length of the longer operand for the comparison.
- If no other relationship can be determined after the strings are exhausted, the strings are equal.

The collating sequence depends partially on the processor; however, equality tests **.EQ.** and **.NE.** do not depend on the processor collating sequence and can be used on any processor.

Logical Expressions

A logical expression specifies a logical computation that yields a logical value. The simplest form of a logical expression is one of the following:

- logical constant
- logical variable reference
- logical array element reference
- logical function reference
- relational expression

Construct complicated logical expressions from one or more logical operands together with logical operators and parentheses.

Logical Operators

Table 3−5defines the Fortran logical operators.

Table 3−5 Logical Operators

Logical Operator	Meaning
.NOT.	Logical negation
.AND.	Logical conjunt
.OR.	Logical disjunct
.EQV.	Logical equivalence
.NEQV.	Logical exclusive or
.XOR.	Same as .NEQV.

All logical operators require at least two operands, except the logical negation operator**.NOT.** , which requires only one.

A logical expression containing two or more logical operators is evaluated based on a precedence relation between the logical operators. This precedence, from highest to lowest, is

- **.NOT.**
- **.AND.**
- **.OR.**
- **.EQV.** and **.NEQV.**
- **.XOR.**

For example, in the following expression

```
W .NEQV. X .OR. Y .AND. Z
```

the operators are executed in the following sequence:

1. **Y .AND. Z** is evaluated first (**A** represents the result).
2. **X .OR. A** is evaluated second (**B** represents the result).
3. **W .NEQV. B** is evaluated to produce the final result.

You can use parentheses to override the precedence of the operators.

Logical Operands

Logical operands specify values with a logical data type. The forms of a logical operands are

- logical primary
- logical factor
- logical term
- logical disjunct
- logical expression

Logical Primary

The logical *primary* is the basic component of a logical expression. The forms of a logical primary are

- logical constant
- symbolic name of a logical constant
- integer or logical variable reference
- logical array element reference
- integer or logical function reference
- relational expression
- integer or logical expression in parentheses

When an integer appears as an operand to a logical operator, the other operand is promoted to type integer if necessary and the operation is performed on a bit−by−bit basis producing an integer result. Whenever an arithmetic datum appears in a logical expression, the result of that expression will be of type integer because of type promotion rules. If necessary, the result can be converted back to **LOGICAL**.

Do not specify two logical operators consecutively and do not use implied logical operators.

Logical Factor

The *logical factor* uses the logical negation operator **.NOT.** to reverse the logical value to which it is applied. For example, applying **.NOT.** to a false relational expression makes the expression true. Therefore, if **UP** is true, **.NOT. UP** is false. The logical factor has the following forms:

- logical primary
- **.NOT.** logical primary

Logical Term

The *logical term* uses the logical conjunct operator **.AND.** to combine logical factors. It takes the forms

- Logical factor
- Logical term **.AND.** logical factor

In evaluating a logical term with two or more **.AND.** operators, the logical factors are combined from left to right. For example, **X .AND. Y .AND. Z** has the same interpretation as **(X .AND. Y) .AND. Z**.

Logical Disjunct

The *logical disjunct* is a sequence of logical terms separated by the **.OR.** operator and has the following two forms:

- Logical term

- Logical disjunct **.OR.** logical term

In an expression containing two or more **.OR.** operators, the logical terms are combined from left to right in succession. For example, the expression **X .OR. Y .OR. Z** has the same interpretation as **(X .OR. Y) .OR. Z**.

Logical Expression

At the highest level of complexity is the *logical expression.* A logical expression is a sequence of logical disjuncts separated by the **.EQV**., **.NEQV.**, or **.XOR.** operators. Its forms are

- logical disjunct
- logical expression **.EQV.** logical disjunct
- logical expression **.NEQV.** logical disjunct
- logical expression **.XOR.** logical disjunct

The logical disjuncts are combined from left to right when a logical expression contains two or more **.EQV.**, **.NEVQ.**, or **.XOR.** operators.

A logical constant expression is a logical expression in which each primary is a logical constant, the symbolic name of a logical constant, a relational expression in which each primary is a constant, or a logical constant expression enclosed in parentheses. A logical constant expression can contain arithmetic and character constant expressions but not variables, array elements, or function references.

Interpretation of Logical Expressions

In general, logical expressions containing two or more logical operators are executed according to the hierarchy of operators described previously, unless the order has been overridden by the use of parentheses. Table 3−6defines the form and interpretation of the logical expressions.

Table 3−6 Logical Expressions

IFA=	B=	THEN .NOT.B	A.AND.B	A.OR.B	A.EQV.B	A.XOR.B A.NEQV.B
F	F	T	F	F	T	F
F	T	F	F	T	F	T
T	F	-	F	T	F	T
T	T	-	T	T	T	F

Evaluating Expressions in General

Several rules are applied to the general evaluation of Fortran expressions. This section covers the priority of the different Fortran operators, the use of parentheses in specifying the order of evaluation, and the rules for combining operators with operands.

Note: Any variable, array element, function, or character substring in an expression must be defined with a value of the correct type at the time it is referenced.

Precedence of Operators

Certain Fortran operators have precedence over others when combined in an expression. The previous sections have listed the precedence among the arithmetic, logical, and expression operators. No precedence exists between the relational operators and the single character operator (//). On the highest level, the precedence among the types of expression operators, from highest to lowest, is

- arithmetic
- character
- relational
- logical

Integrity of Parentheses and Interpretation Rules

Use parentheses to specify the order in which operators are evaluated within an expression. Expressions within parentheses are treated as an entity.

In an expression containing more than one operation, the processor first evaluates any expressions within parentheses. Subexpressions within parentheses are evaluated beginning with the innermost subexpression and proceeding sequentially to the outermost. The processor then scans the expression from left to right and performs the operations according to the operator precedence described previously.