4. EXPRESSIONS

4.1. GENERAL

An expression is a rule for computing a value. There are four kinds of expressions: arithmetic, Boolean, string, and designational. Expressions are composed of operands, operators and parentheses. Operands are constants, variables, function designators, or other expressions. Operators are symbols which designate arithmetic, relational, or logical operations, and parentheses are used to determine the sequence of operations to be performed. The value of an arithmetic expression is a number of the type **INTEGER, REAL, REAL 2** or **COMPLEX.** The value of a Boolean expression is either **TRUE** or **FALSE**, and the value of a string expression is a string of symbols. The value of a designational expression is a statement label. Expressions must be formed in accordance with mathematical convention and with the rules discussed in the following paragraphs.

4.2. ARITHMETIC EXPRESSIONS

Arithmetic quantities are combined into arithmetic expressions by means of the following arithmetic operators:

+ denotes addition
– denotes subtraction
* denotes multiplication
/ denotes division
** denotes exponentiation
// denotes integer division

The expression

A//B

can be written only when A and B are both of type **INTEGER.** The expression has the integer value of the unrounded quotient of A by B.

Thus 5//3 = 1

The expression

A**B

means A raised to the power B.

4.2.1. Ordering Rules for Operations

The hierarchy of arithmetic operations is

Class 1	**	Exponentiation
Class 2	*	Multiplication
	/	Division
	//	Integer division
Class 3	+	Addition
	–	Subtraction

Expressions with operators in different classes are evaluated in order (1,2, and then 3) unless parentheses are used to change the order. Expressions containing operators in the same class are evaluated from left to right. Parentheses may be used to override the given order of evaluation. Expressions within innermost parentheses are evaluated first.

Examples:

Expression	Compiler Interpretation
A-B-C	(A-B)-C
A-B**C	A-(B**C)
A**B-C**D	(A**B)-(C**D)
A+B/C	A+(B/C)
A/B/C	(A/B)/C
A**B**C	(A**B)**C

4.2.2. Hierarchy of Operand Types

Mixed-mode arithmetic is permitted. The evaluated arithmetic expression assumes the type of the dominant operand type in the expression. The order of dominance is **COMPLEX**, **REAL 2**, **REAL**, and **INTEGER**. Exponentiation routines for **COMPLEX** ****REAL 2** and **REAL 2****COMPLEX** have not been implemented.

Example:

```
INTEGER    I $
REAL       R $
REAL 2     R2 $
COMPLEX    C $
```

then

```
I*R        IS REAL
R2+R       IS REAL 2
C-R2+I     IS COMPLEX
```

There are two exceptions to the above rule:

- A/B is **REAL** when A and B are **INTEGER**
- A**B is **REAL** when A and B are **INTEGER**

4.2.3. Operands of Arithmetic Expressions

The operands of arithmetic expressions are constants, variables, function designators (defined below), or other arithmetic expressions.

4.2.3.1. Subscripted Variables

A variable may be either a simple variable or a subscripted variable. A subscripted variable represents one of the following:

(1) A single element of an array denoted by the identifier which names the array followed by a subscript list enclosed in parentheses,

(2) A portion of a string variable, or

(3) A combination of both (1) and (2) in the case of **STRING** arrays. A subscript list consists of arithmetic expressions separated by commas.

The following are examples of subscripted variables:

```
A(I,J)
M(I+1,J+1)
V(F(P+1),Q+12)
Z(W(T),X(T),Y(T),Z(T))
X(13)
A(I*2,I//2)
```

The expressions which make up the subscript may be of any complexity. **REAL** values are allowed, in which case the real number is rounded to the nearest integer. Each subscript expression must have a value which is not less than the minimum and not greater than the maximum specified by the **ARRAY** declaration or for the string as specified by the **STRING** declaration (see Section 3). The number of subscript expressions must equal the number of dimensions of the array as given in the **ARRAY** declaration. Thus, if an array is declared as follows,

```
REAL ARRAY A(1:10,1:10)
```

then A(3,11) is undefined.

4.2.3.2. Function Designators

A function designator is either a call on a declared function procedure (see Section 7) or a call on a standard function. These standard functions are the ones commonly employed in mathematics, such as the square root, sine, and arctangent functions. A complete list of the available functions is given in Appendix B. For example, the function whose value is the square root of X is called SQRT; therefore if

```
REAL X ;
```

then

```
SQRT (X)
```

is a function designator.

Operands themselves may be arithmetic expressions, and combining them by means of operators may give rise to more arithmetic expressions. Assuming the declarations:

```
REAL R $
INTEGER I $
INTEGER ARRAY A(0:10) $
```

Then the following are valid arithmetic expressions:

```
(R*I)/(A(1)+7)
(A(A(2))-I**3)*MOD(A(7),4)
```

MOD is an example of a standard function. In these examples, liberal use is made of parentheses to indicate order of evaluation.

4.3. BOOLEAN EXPRESSIONS

The only Boolean constants are **TRUE** and **FALSE** and these have their fixed, obvious meaning. A Boolean operand may be either a simple variable that has occured in a Boolean declaration, a subscripted variable that has appeared in a Boolean array declaration or a Boolean function designator such as NUMERIC, or a Boolean constant (see Appendix B). Boolean expressions are:

- Boolean operands
- Arithmetic or string expressions connected by the relational operators **LSS, LEQ, EQL, GEQ, GTR** or **NEQ**
- Boolean expressions connected by the logical operators **NOT, AND, OR, IMPL, EQIV,** or **XOR**

4.3.1. Relational Expressions

The relational operators have the semantic meanings

ALGOL EXPRESSION	MATHEMATICAL NOTATION	MEANING
A **LSS** B	$A < B$	Less than
A **LEQ** B	$A \leq B$	Less than or equal to
A **EQL** B	$A = B$	Equal to
A **GEQ** B	$A \geq B$	Greater than or equal to
A **GTR** B	$A > B$	Greater than
A **NEQ** B	$A \neq B$	Not equal to

For arithmetic or string expressions A and B, the Boolean expression:

A<relational operator>B

is **TRUE** if the relation holds and **FALSE** if it does not. A and B may be mixed mode. If either A or B is **COMPLEX**, only the relations **EQL** and **NEQ** can be used. If A and B are both string expressions (see 4.4), the strings are compared character-by-character starting at the left. The shorter string is considered to be filled out with blanks to the length of the longer. The collation sequence of characters is that of Fieldata.

4.3.2. Boolean Operators

The six Boolean (logical) operators are:

NOT	negation
AND	conjunction
OR	inclusive disjunction (inclusive OR)
IMPL	implication
EQIV	equivalence
XOR	exclusive disjunction (exclusive OR)

The value of a Boolean expression of the form:

A<Boolean operator>B

is obtained from the following table. **NOT** is a unary operator.

A	B	**NOT** A	A **OR** B	A **AND** B	A **XOR** B	A **IMPL** B	A **EQIV** B
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE

Assume the following declarations:

```
BOOLEAN A,B $
REAL X,Y,Z $
STRING S(100) $
```

Then the following are legitimate Boolean expressions.

```
B AND A
10.0 LEQ X AND X LEQ 99.0
NOT B OR A
NOT (X LSS Y IMPL Z EQL Z**2)
NUMERIC(S) OR NOT ALPHABETIC(S)
```

In the above example NUMERIC and ALPHABETIC are standard functions (see Appendix B).

4.3.3. Precedence of Boolean Operations

Parentheses may be used to specify the order of operations in Boolean expressions. If parentheses are omitted (or within parentheses), Boolean expressions are scanned from left to right, and operations are performed according to the following precedence:

(1) Arithmetic operations according to 4.2.1.

(2) Relational operations

(3) **NOT**

(4) **AND**

(5) **OR XOR**

(6) **IMPL**

(7) **EQIV**

Example:

```
A+1 GTR B OR C AND G+2 NEQ H
```

will be computed as

```
((A+1) GTR B) OR (C AND ((G+2) NEQ H))
```

Parentheses should be used in compound expressions to avoid confusion or misinterpretation by human readers (not the compiler).

4.4. STRING EXPRESSIONS

Strings have no operators which produce a string result. Substrings of a declared string are defined by giving a starting position and a length. For example, if S is a string variable

```
STRING S(120)
```

then

S(i)

denotes the ith character in the string S where the characters are numbered from the left starting with one. Thus, S(6) is the sixth character in string S.

S(i,j)

denotes a string of j characters taken in ascending order from string S starting with the character in the ith position. S(1,10) is the substring of S consisting of the first ten characters of S. The substring S(1) is equivalent to S(1, 1). That is, if the length is omitted, it is taken to be 1.

In summary, consider string S.

```
STRING S(10,R(20),STACK(3,OP(5)),4)
```

In this string the following three string expressions all have the same value; that is, they all refer to the same five characters.

```
S(34,5)
STACK (4,5)
OP
```

Partial string-array variables are subscripted variables with two separate subscript lists separated by a colon (:). A subscripted string variable is written as

S(<string part>:<subscript list>)

If S is a two-dimensional string array, then

S(i:j, k)

denotes the ith character in the j, kth element of the string array S. If a group of characters is desired, then

S(i, 1:j, k)

will denote the group of 1 characters taken in ascending order starting with the character in the ith position from the j, kth element in the string array. On the other hand, if the entire string from the j, kth element in the string array is desired, then the colon may be omitted. Thus,

S(j, k)

specifies the entire string.

Example:

```
STRING ARRAY S(L(40),R(40):1:10,1:10)
```

then

```
S(41,40:4,10)
```

specifies the last 40 characters of the string in the fourth row and tenth column of the two-dimensional array. Each of the elements consists of 80 characters.

A numeric string expression may be used as an arithmetic operand:

```
S(1,5)+1
S(1) EQL 1
```

When used in this context, the string expression is converted to an integer expression by a transfer function. If the string does not represent an integer, an error message is printed (see Appendices B and C). If the integer value of the string is greater than $(2^{35}-1)$, an error message results.

4.5. DESIGNATIONAL EXPRESSIONS

Designational expressions are expressions whose values are statement labels (see 5.5). The form of a designational expression is either a label, a switch variable, or a conditional expression in which the value of a Boolean expression determines which of two designational expressions to use. For further details see 6.2 and 6.3.

4.6. CONDITIONAL EXPRESSIONS

The value of an expression may depend on the value of a Boolean expression. For example,

```
IF X EQL Y THEN 1 ELSE 2
```

is an integer expression whose value is 1 or 2 depending on whether X equals Y. The general form of a conditional expression is:

IF <Boolean expression> **THEN** <simple expression>
ELSE <expression>

The expression following **THEN** and the expression following **ELSE** must be of the same kind (arithmetic, Boolean, or string). The expression following **ELSE** may be another conditional expression.

Example:

```
IF X GTR 0 THEN 0 ELSE (X EQL 0)
```

is illegal because in some cases it has an arithmetic value (0) and in other cases a Boolean value (X **EQL** 0). In the cases where the constituent expressions are arithmetic, then the type of the entire expression is always the more general of the two expressions:

```
IF X EQL 0 THEN <1.0,2.0> ELSE 2.0
```

has a value of either <1.0,2.0> or <2.0,0.0>; that is, a value of type **COMPLEX**. Note that **ELSE** must always be present in conditional expressions.

The <simple expression> may be any expression not starting with **IF**, or any expression put into parentheses. For example, the following is not a simple expression because it begins with **IF**, but it contains a simple expression in the parentheses.

```
IF A THEN X+(IF B THEN X ELSE Z) ELSE IF B THEN Z ELSE 0
```