How do you write Fortran expressions?

Products: Aspen Plus
Last Updated: 03-Nov-2020
Versions: V14.3, V14.2, V14.0, V12.1, V12.0, V11.0, V10.0, V9.0, V8.8, V8.6
Article ID: 000086576
Primary Subject: Flowsheet & Model Analysis

Problem Statement

How do you write Fortran expressions?

Solution

We are adding a new help topic with some simple instructions on how to use Fortran since so many new engineers have no experience with it. It is important to note that Fortran expressions are used not only in the Calculator block, but on other forms such as Design specification and Sensitivity. Also, basic Fortran is no more complicated than writing an equation in Excel.

How to Write Fortran Expressions

Not sure how to use Fortran? This topic explains the basics of writing code to perform simple calculations in Fortran.

Fortran used in Aspen Plus is limited to Fortran 77 syntax, which the following sections describe. Fortran variable names and function names are not case sensitive; **PRES**, **Pres**, and **pres** all refer to the same variable.

All of the syntax described on this page can be interpreted, which means that you do not need to have a Fortran compiler installed to use it. Some other Fortran statements can also be interpreted; for a full list, see About the Interpreter.

Assignments and Arithmetic Operators

The most commonly used Fortran statements are assignment statements, which have the form

variable = Fortran expression

The variable can be one you have defined as a variable in a Calculator block, or one defined in a declaration statement. The Fortran expression can be a number, another variable, a function call, or an arithmetic operation combining two or more expressions of these types. The value resulting from evaluating the Fortran expression is assigned to the variable when this statement is executed. In some cases in Aspen Plus, such as the Spec, Target, and Tolerance of a Design-Spec and the limits on manipulated variables, you will enter *only* a Fortran expression. In this case, there is an implicit assignment to the indicated attribute (in the same way that you would simply enter a number in most fields), but the expression is evaluated each time the value of the attribute is required.

The basic arithmetic operators are:

Addition

The + sign is used for addition. The following statement adds 1 to the value of B and assigns the result to A:

```
A = B + 1
```

Subtraction

The - sign (ASCII hyphen, not an em dash or en dash character) is used for subtraction. The following statement subtracts 1 from the value of B and assigns the result to A:

```
A = B - 1
```

Multiplication

The * character (asterisk) is used to represent multiplication. The following statement multiplies the value of B by 2 and assigns the result to A:

```
A = B * 2
```

Division

The / character (slash) is used to represent division. The following statement divides the value of B by 2 and assigns the result to A.

```
A = B / 2
```

Note: Division of one integer-type variable by another is integer division: the numbers are divided, the whole part of the result is kept (as an integer-type value), and the remainder is ignored.

Exponentiation

Two asterisks (**) are used to represent exponentiation. Keep in mind the standard mathematical restrictions on exponentiation. The following statement squares the value of B and assigns the result to A:

```
A = B ** 2
```

Order of Operations

You can combine multiple arithmetic operations into a single expression. When you do so, Fortran has a specific order in which it performs the operations:

- Exponentiations are performed first, right to left.
- Multiplications and divisions are performed next, left to right.
- Addition and subtraction is performed last, left to right.

For example, 2+5*3 evaluates to 17 because the product 5*3 is calculated first, and then 2 is added to it.

You can group expressions in parentheses to specify a different order of calculations. Everything in parentheses is evaluated before anything outside the parentheses. If parentheses are nested, operations inside the inner parentheses are performed before those in the outer parentheses. For example, (2+5)*3 evaluates to 21.

Comments, Line Numbers, Continuation, and Indentation

You may notice how the examples in this topic are all indented. In Fortran statements, the first 6 columns are special.

- A comment line, which is ignored during calculations, can be indicated by placing a C or c in the first column.

- Line numbers can be indicated by writing the numbers into the third, fourth, and/or fifth columns. These can be used in certain kinds of statements to refer to another line. In lines which are neither comments nor numbered, you should leave the first five columns blank (spaces).

- The sixth column is used only for the continuation character, to indicate that the line is a continuation of the previous line when expressions are very long. Fortran lines must not be longer than 72 characters, including the initial 6 spaces. Any character other than a space in the sixth column will make the line a continuation, but it is traditional to use a plus sign or to use digits (2 for the second line, 3 for the third line, and so forth).

The built-in editor in the **Calculator | Input | Calculate** sheet automatically leaves 6 blank spaces at the start of each line. If you need to make comments, line numbers, or continuations, you can delete these spaces.

Variable Types and Declarations

Fortran variables have explicit types indicating the kind of data they can hold. The most common variable types found in Fortran used within Aspen Plus are:

- Integer: A variable that holds a whole number such as 0, 1, or -2. Integers are stored in 4 bytes or 32 bits, one of which is used to store the sign, so they can hold values between 2^{31} and -(2^{31}), or about 2,000,000,000 and -2,000,000,000. Note, though, that you cannot enter commas when writing large numbers into your Fortran program; just write **10000** instead of 10,000.

- Real*8: A real variable which can hold a whole or fractional (decimal) value. The *8 indicates that the variable uses 8 bytes or 64 bits. This kind of variable (also called *double precision*) can store about 14 digits of accuracy and can store numbers up to about 10^{308} or as small as 10^{-308}, as well as the negatives of this range. When you enter decimal numbers directly into your Fortran program, be sure to use a period or full-stop (.) as the decimal separator, even if Aspen Plus is configured to use a comma for the decimal separator elsewhere. For example, **1.5** is the correct way to write the number one-and-a-half.

Two less commonly used types are:

- Character*n: A character variable can store a string of text. The n indicates the maximum length of the string which the variable can hold.

- Logical: A variable which can store a true or false value. If you want to write literal true and false values in your Fortran program, they have to be enclosed in period or full-stop characters (.TRUE. or .FALSE.). You can also generate Logical data as the result of using comparison operators.

When you perform arithmetic operations on two numbers, the result is the same type as the numbers you are operating on. If one number is real*8 and the other is integer, the result will be real*8.

Variables you define with a reference to an Aspen Plus variable are automatically declared with the type appropriate to the Aspen Plus variable. This is usually Real*8, but countable items such as stage numbers are of type Integer. If you want to make intermediate variables used during calculations, enter declarations for these variables in the **Fortran Declarations** dialog box of a Calculator block, or at the start of an external Fortran subroutine. Declarations consist of the variable type followed by the variable, separated by a space. You can declare multiple variables of the same type by separating them with commas. To declare an array variable, enter the dimensions in parentheses, with multiple dimensions separated by commas, after the variable name in a declaration. Example declarations:

```
INTEGER I, J(2)

REAL*8 TIME,VOLUME,PRES(7,10)

LOGICAL C

CHARACTER*10 NAME
```

A Fortran variable name must:

- Be eight characters or less
- Start with an alphabetic character (A – Z)
- Have subsequent alphanumeric characters (A – Z, 0 – 9)
- Not begin with IZ or ZZ

Conditions and Branching

You can write IF statements to perform certain operations only in certain conditions. The format of an IF statement is:

IF (*logical expression*) THEN

  conditional code

END IF

The logical expression can be a logical variable or the result of a comparison or logical operator. The parentheses around the logical expression are required. If the logical expression evaluates to true, then the conditional code is executed, and otherwise it is skipped.

Comparison Operators

Comparison operators can be used in decision statements, or to store a value in a logical variable which may be used in a decision statement later. The comparison operators in Fortran are:

Operator	Meaning

.LT.	Less than
.GT.	Greater than
.EQ.	Equal
.LE.	Less than or equal
.GE.	Greater than or equal
.NE.	Not equal

For example, the expression B .LT. 3 is true if B is less than 3.

Logical Operators

For complicated logical expressions you can use logical operators to combine multiple logical expressions. The logical operators in Fortran are:

Operator	Meaning
.AND.	And (true only if both expressions are true)
.OR.	Or (true if either or both expressions are true)
.NOT.	Not (reverses result of following logical expression)

For example, the expression A.EQ.3 .OR. B.LT.2 is true whenever A equals 3, B is less than 2, or both.

You can use parentheses to group parts of expressions involving comparison and logical operators. All arithmetic operations are performed first, then comparisons, and logical operators are last, with .NOT. evaluated before .AND., then .OR. is evaluated last.

Function Calls

You can call functions by typing the function name followed by its arguments in parentheses. If the function takes more than one argument, separate the arguments with commas (,).

Most commonly you will call the following built-in Fortran functions (the ones beginning with D return a double precision or real*8 result, while the others return an integer result):

Function	Meaning
DABS(X), IABS(J)	Absolute value
DSIN(X), DCOS(X), DTAN(X)	Sine, cosine, and tangent functions of X in radians.
DASIN(X), DACOS(X), DATAN(X)	Inverse sine, cosine, and tangent functions, with the result returned in radians.
DEXP(X)	Exponential function (e^X)
DLOG(X)	Natural logarithm of X
DLOG10(X)	Base 10 logarithm of X
DSQRT(X)	Square root of X
DMIN1(X1,X2,...), DMAX1(...)	Minimum and maximum of the arguments (two or more real*8 arguments)
MIN0(J1,J2,...), MAX0(...)	Minimum and maximum of the arguments (two or more integer arguments)
DFLOAT(J)	Converts an integer value to a real*8 value
IDINT(X)	Converts a real value to an integer, truncating the fractional part. IDINT(1.3)=1; IDINT(-2.7)=-2.

-