Data Types and Declarations

Data Types

FORTRAN supports six data types:

CHARACTER
  character

COMPLEX
  single precision complex number

DOUBLE PRECISION
  double precision floating point number

INTEGER
  integer

LOGICAL
  boolean (true or false)

REAL
  single precision floating point number

Numerical data in FORTRAN can be represented in one of four types. An INTEGER is any signed number that has no fractional part and no decimal point.

-13
12345
+5

INTEGER numbers are also referred to as fixed point numbers. The other three numerical types are called floating point numbers. A REAL number is a signed number with a decimal point.

-13.0
123.45
+0.0005

Very large or very small numbers are often represented in scientific notation. In this notation, a number is represented as

\[ \pm b \times 10^{\pm n} \]

where \( b \) is a number between 1 and 10 and \( n \) is the appropriate power of ten. FORTRAN offers a similar representation called exponential notation:

\[ \pm 0.mE\pm p \]
In this case, the mantissa $m$ is a number between 0.1 and 1.0 and $p$ is again the appropriate power of ten.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Scientific</th>
<th>Exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000135</td>
<td>$1.35 \times 10^{-4}$</td>
<td>$0.135E-03$</td>
</tr>
<tr>
<td>-246.8</td>
<td>$-2.468 \times 10^{2}$</td>
<td>$-0.2468E+03$</td>
</tr>
<tr>
<td>2357000000000000000.0</td>
<td>$2.357 \times 10^{20}$</td>
<td>$0.2357E+21$</td>
</tr>
</tbody>
</table>

A number stored in a computer is limited in magnitude and precision. The limits depend on the particular computer. Thus, a REAL number has only a certain number of significant digits. If more significant digits are required for a calculation, then DOUBLE PRECISION numbers must be used. A DOUBLE PRECISION constant is written in the same exponential form as a single precision REAL constant except with a D instead of an E separating the mantissa from the exponent.

In practice, most computers use 32 bits to store INTEGER and REAL numbers. This means that an INTEGER is limited to numbers between -2,147,483,648 and +2,147,483,647 (a sign bit and 31 magnitude bits). If the IEEE standard is used, then a REAL number will have about seven decimal digits and be within the magnitude range of $10^{-38}$ to $10^{+38}$. DOUBLE PRECISION numbers usually have at least twice the number of significant decimal digits but may have the same magnitude range of REAL numbers.

Complex numbers are common in many fields of science and engineering so it is not surprising that FORTRAN offers a COMPLEX data type. The complex number $a + ib$ where $i$ is the imaginary unit (square root of -1) is represented in FORTRAN as $(a,b)$ where $a$ and $b$ themselves are single precision REAL numbers. There are no double precision complex numbers available in FORTRAN.

The other two data types deal with non-numerical information. A LOGICAL value is either .TRUE. or .FALSE. (note the full stops!) whilst a CHARACTER value can contain any combination of characters from the FORTRAN character set. In fact, on most computers, a CHARACTER string can contain any combination of printable characters. A CHARACTER constant is any set of characters enclosed in apostrophes. If an apostrophe is needed as part of the string, then two apostrophes (not a double quote) are used.

<table>
<thead>
<tr>
<th>CHARACTER Constant</th>
<th>Result</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a character string'</td>
<td>a character string</td>
<td>18</td>
</tr>
<tr>
<td>'Let's go!'</td>
<td>Let's go!</td>
<td>9</td>
</tr>
<tr>
<td>'$ 10.25'</td>
<td>$ 10.25</td>
<td>9</td>
</tr>
<tr>
<td>' '</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**Type Declarations**
The type of any constant, variable or array used in a FORTRAN program must be specified either implicitly or explicitly. In implicit typing, all constants, variables and arrays beginning with the letters I, J, K, L, M, or N are automatically taken to be of type INTEGER. Constants, variables and arrays beginning with all other letters are assumed to be REAL. Thus, with implicit typing, the variable COUNT is REAL whilst the variable KOUNT is an INTEGER. Implicit typing can be overridden with explicit type declaration. To explicitly declare a constant, variable or array to be of a given type, simply put a statement of the form

\[ \text{type name-list} \]

where type is one of the six data types and the name-list is a list of the names of the constants, variables or arrays of the chosen type separated by commas.

The declarations

```
COMPLEX     FALL,TRIP
DOUBLE PRECISION BIGJMP
INTEGER     A,AA,AAA
LOGICAL     DECIDE
REAL        BOUND,JUMP,LEAP
```

at the beginning of a program unit define the variables named FALL and TRIP to be of type COMPLEX; BIGJMP to be DOUBLE PRECISION; A, AA, and AAA to be INTEGER; DECIDE to be LOGICAL; and BOUND, JUMP, and LEAP to be REAL.

CHARACTER declarations are a little more subtle in that they require prior knowledge of the length of the string that will be stored in the CHARACTER variable. The syntax for a CHARACTER variable is

\[ \text{CHARACTER}^*m \text{ variable-list} \]

where all of the variables in the variable-list are \( m \) characters long. Some or all of these characters may be blank. It is also possible to use one declaration statement to specify several variables of different lengths:

```
CHARACTER variable_1^*m_1, variable_2^*m_2, ..., variable_n^*m_n
```

In this case, the first variable is of length \( m_1 \), the second variable is of length \( m_2 \) and so on.

The declarations

```
CHARACTER*3 CONST,GREEK
CHARACTER   CATLOG^*10,NAME^*20
```
at the beginning of a program unit define **CONST** and **GREEK** to be **CHARACTER** variables of length 3 whilst **CATLOG** is of length 10 and **NAME** is of length 20.

**IMPLICIT Statement**

Although only **INTEGER** and **REAL** constants, variables and arrays have implicit types, it is possible to assign defaults for all data types through the use of the **IMPLICIT** statement. The syntax for this statement is

**IMPLICIT** *type$_1$ (range$_1$), type$_2$ (range$_2$), ..., type$_n$ (range$_n$)*

Explicit type declarations override implicit type declarations.

A program that has only **DOUBLE PRECISION** variables might contain the statement

**IMPLICIT DOUBLE PRECISION(A-Z)**

at the beginning.

Because of the declarations

**IMPLICIT COMPLEX(A-C),DOUBLE PRECISION(D),INTEGER(E-Z)**

**LOGICAL** **HELP**

all variables beginning with the letters A through C are of type **COMPLEX**, all variables beginning with the letter D are of type **DOUBLE PRECISION**, and everything else is an **INTEGER**. The explicit type declaration that **HELP** is of type **LOGICAL** overrides the **INTEGER** default.

**PARAMETER Statement**

A **PARAMETER** statement is used to assign a constant value to a symbolic name.

**PARAMETER**(cname$_1$ = value$_1$, cname$_2$ = value$_2$, ..., cname$_n$ = value$_n$)

Although the value of a FORTRAN constant cannot be changed elsewhere in the program, it can be used in other **PARAMETER** statements as well as in type declarations, **DATA** statements and in calculations.

**DOUBLE PRECISION DEG,PI,RAD,TWOPI**

**PARAMETER**(PI=3.141592653589793D0,TWOPI=2D0*PI)
The value of the DOUBLE PRECISION constant \( \text{PI} \) is assigned in the PARAMETER statement. The value of another DOUBLE PRECISION constant, \( \text{TWOPI} \), is also assigned in the same PARAMETER statement. This is done simply by multiplying the already-defined constant \( \text{PI} \) by \( 2\cdot10^0 \). Later in the program, the value of \( \text{PI} \) is used in an arithmetic expression.

```
INTEGER   COLS, ROWS
PARAMETER(ROWS=12, COLS=10)
REAL      MATRIX(ROWS, COLS), VECTOR(ROWS)
```

In this set of declarations, the constants \( \text{COLS} \) and \( \text{ROWS} \) are declared to be of type INTEGER before being given the values in the following PARAMETER statement. After the constants have been defined, they can be used in the array declarations on the next line. Using named constants are array bounds is a common application in FORTRAN.

CHARACTER constants are also possible. They can be declared in the following manner:

```
CHARACTER(*) \( \text{cname} \)
PARAMETER(\( \text{cname} = '\text{string}' \))
```

The length of the CHARACTER constant \( \text{cname} \) is automatically set to the length of the \text{string} in the PARAMETER statement. This only works with CHARACTER constants, not CHARACTER variables which must have the length explicitly declared. The CHARACTER*(*) statement must precede the PARAMETER statement.

```
CHARACTER(*) \( \text{ERRMSG} \)
PARAMETER(\( \text{ERRMSG} = '\text{Division by zero!}' \))
```

In this example, the CHARACTER constant \( \text{ERRMSG} \) is 17 characters long and contains the string Division by zero!

LOGICAL constants can take the values .TRUE. or .FALSE.

```
LOGICAL \( \text{DEBUG} \)
PARAMETER(\( \text{DEBUG} = .\text{TRUE.} \))
```

```
IF (\( \text{DEBUG} \)) WRITE(*,*)'Entering first DO loop'
```

...
The **LOGICAL** constant `DEBUG` is set to `.TRUE.` and is used in an **IF** statement later in the program.