Chapter 5

Assignment and Data Statements

This chapter contains the following subsections:

- "Arithmetic Assignment Statements"
- "Logical Assignment Statements"
- "Character Assignment"
- "Aggregate Assignment"
- "ASSIGN"
- "Data Initialization"
- "Implied−DO Lists"

Assignment statements assign values to variables and array elements. Data statements and implied−**DO** lists in data statements are used to initialize variables and array elements.

The five types of Fortran assignment statements are

- arithmetic
- logical
- character
- aggregate
- statement label

This chapter explains how to use each of these statements. Each type is discussed in detail in the following sections.

Arithmetic Assignment Statements

An arithmetic assignment statement assigns the value of an arithmetic expression to a variable or array element of type **INTEGER**, **REAL**, **DOUBLE PRECISION**, **COMPLEX**, or **DOUBLE COMPLEX**. The form of an arithmetic statement is

v = *e*

where

v	is the name of an integer, real, double−precision, complex, or double−complex type variable or array element.
e	is an arithmetic expression.

When an arithmetic assignment statement is executed, the expression *e* is evaluated and the value obtained replaces the value of the entity to the left of the equal sign.

The values *v* and *e* need not be of the same type; the value of the expression is converted to the type of the variable or array element specified. Table 5−1lists the type conversion rules.

Table 5−1 Type Conversion Rules

Declaration	Function Equivalent
INTEGER	INT(*e*)
REAL	REAL(*e*)
DOUBLE PRECISION	DBLE(*e*)
COMPLEX	CMPLX(*e*)
DOUBLE COMPLEX	DCMPLX(*e*)

The following are examples of arithmetic assignment statements:

I = 4	Assign the value 4 to **I**
J = 7	Assign the value 7 to **J**
A = I*J+1	Assign the value 29 to **A**

Table 5−2gives the detailed conversion rules for arithmetic assignment statements. The functions in the second column of the table are intrinsic functions described in Chapter 10, "Statement Functions and Subprograms" and Appendix A, "Intrinsic Functions."

Table 5−2 Conversion Rules for Assignment Statements

Variable or Array Element (*v*)	INTEGER or LOGICAL Expression (*e*)	REAL Expression (*e*)	REAL *8 Expression (*e*)	REAL *16 Expression (*e*)	COMPLEX Expression (*e*)	COMPLEX *16 Expression (*e*)
INTEGER or LOGICAL	Assign *e* to *v*	Truncate *e* to integer and assign to *v*	Truncate *e* to integer and assign to *v*	Truncate *e* to integer and assign to *v*	Truncate real part of *e* to integer and assign to *v*	Truncate real part of *e* to integer and assign to *v*
REAL	Append fraction (.0) to *e* and assign to *v*	Assign *e* to *v*	Assign high−order portion of *e* to *v*; low−order portion of *e* is rounded	Assign high−order part of *e* to *v*; low−order part is rounded	Assign real part of *e* to *v*; imaginary part of *e* not used	Assign high−order part of real part of *e* to *v*; low−order portion of real part *e* is rounded
REAL *8	Append fraction (.0) to *e* and assign to *v*	Assign *e* to high−order portion of *v*; low−order portion of *v* is 0	Assign *e* to *v*	As above	Assign *e* to high−order portion of *v*; low−order portion of *v* is 0	Assign real part of *e* to *v*
REAL *16	As above	As above	As above	As above	As above	As above
COMPLEX	Append fraction to *e* and assign to real part of *v*; imaginary part of *v* is 0.0	Assign *e* to real part of *v*; imaginary part of *v* is 0.0	Assign high−order portion of *e* to real part of *v*; low−order portion of *e* is rounded; imaginary part of *v* is 0.0	Assign high−order portion of *e* to real part of *v*; low−order part is rounded; imaginary part of *v* is 0.0	Assign *e* to *v*	High−order parts of real and imaginary components of *e* are assigned to *v*; low−order parts are rounded
COMPLEX *16	Append fraction to *e* and assign to *v*; imaginary	Assign *e* to high−order portion of real	Assign *e* to real part of *v*; imaginary part is 0.0	As above	Assign *e* to high−order parts of *v*; low−order parts	Assign *e* to *v*

to *v*, imaginary part of *v* is 0.0	portion of real part of *v*, imaginary part of *v* is 0.0	is 0.0		*v*, low−order parts of *v* are 0

Logical Assignment Statements

The logical assignment statement assigns the value of a logical expression to a logical variable or array element. It takes the form

v = *e*

where

v is the name of a logical variable or logical array element.

e is a logical expression.

When a logical assignment statement is executed, the value of the logical expression *e* is evaluated and replaces the value of the logical entity to the left of the equal sign. The value of the logical expression is either true or false.

Character Assignment

The character assignment statement assigns the value of a character expression to a character variable, array element, or substring. The form of a character assignment statement is

v = *e*

where

v is the name of a character variable, array element, or substring.

e is a character expression.

During the execution of a character string assignment statement, the character expression is evaluated and the resultant value replaces the value of the character entity to the left of the equal sign. None of the character positions being defined in *v* can be referenced in the evaluation of the expression *e*.

The entity *v* and character expression *e* can have different lengths. If the length of *v* is greater than the length of *e*, then the value of *e* is extended on the right with blank characters to the length of *v*. If the length of *e* is greater than the length of *v*, then the value of *e* is truncated on the right to the length of *v*.

The following is an example of character assignment:

```
CHARACTER U*5, V*5, W*7
U = 'HELLO'
V = 'THERE'
W(6:7) = V(4:5)
```

If an assignment is made to a character substring, only the specified character positions are defined. The definition status of character positions not specified by the substring remain unchanged.

Aggregate Assignment

An aggregate assignment statement assigns the value of each field of one aggregate to the corresponding field of another aggregate. The aggregates must be declared with the same structure. The form of an aggregate assignment statement is

v = *e*

where *v* and *e* are aggregate references declared with the same structure.

See the "Records" section and "Record and Field References" subsections in Chapter 2, "Constants and Data Structures," for more information.

ASSIGN

The **ASSIGN** statement assigns a statement label to an integer variable and is used in conjunction with an assigned **GOTO** statement or an I/O statement. The form of a statement label assignment statement is

`ASSIGN` *s* `TO` *e*

where

s — is a statement label of an executable statement or a **FORMAT** statement that appears in the same program unit as the **ASSIGN** statement.

e — is an integer variable name.

A statement label assignment by the **ASSIGN** statement is the only way of defining a variable with a statement label value. A variable defined with a statement label value may be used only in an assigned **GOTO** statement or as a format identifier in an I/O statement. The variable thus defined must not be referenced in any other way until it has been reassigned with an arithmetic value.

An integer variable that has been assigned a statement label value can be redefined with the same statement label, a different statement label, or an arithmetic integer variable.

Examples using the **ASSIGN** statement are shown below:

Example 1

```
ASSIGN 100 TO kbranch
    .
    .
    .
GO TO kbranch
```

Example 2

```
ASSIGN 999 TO ifmt
999 FORMAT(f10.5)
    .
    .
```

```
     .
READ (*, ifmt) x
     .
     .
     .
WRITE (*, fmt = ifmt) z
```

Data Initialization

Variables, arrays, array elements, and substrings can be initially defined using the **DATA** statement or an implied-**DO** list in a **DATA** statement. The **BLOCK DATA** subprogram is a means of initializing variables and arrays in named common blocks and is discussed in Chapter 4, "Specification Statements."

Entities not initially defined or associated with an initialized entity are undefined at the beginning of the execution of a program. Uninitialized entities must be defined before they can be referenced in the program.

Implied−DO Lists

The implied-**DO** list initializes or assigns initial values to elements of an array.

Syntax

(*dlist*, *i* = *e1*, *e2* [,*e3*])

where

dlist	is a list of array element names and implied-**DO** lists.
i	is a name of an integer variable, referred to as the *implied−DO variable* It is used as a control variable for the iteration count.
e1	is an integer constant expression specifying an initial value.
e2	is an integer constant expression specifying a limit value.
e3	is an integer constant expression specifying an increment value.

e1, *e2*, and *e3* are as defined in **DO** statements.

Method of Operation

An iteration count and the values of the implied-**DO** variable are established from *e1*, *e2*, and *e3* exactly as for a **DO**-loop, except that the iteration count must be positive.

When an implied-**DO** list appears in a **DATA** statement, the *dlist* items are specified once for each iteration of the implied-**DO** list with the appropriate substitution of values for any occurrence of the implied-**DO** variable. The appearance of an implied-**DO** variable in an implied-**DO** has no effect on the definition status of that variable name elsewhere in the program unit. For an example of an implied-**DO** list, see "DATA" in Chapter 4.

The *range* of an implied-**DO** list is *dlist*.

Rules

- The integer constant expressions used for *e1*, *e2*, and *e3* can contain implied-**DO** variables of other implied-**DO** lists.
- Any subscript expression in the list *dlist* must be an integer constant expression. The integer constant expression can contain implied-**DO** variables of implied-**DO** lists that have the subscript expression within their range.