Chapter 9

Data Initialization

The initial value of a variable can be specified in its declaration using the DATA attribute, as described in Section 5.4. An alternative is to use a DATA statement in a program unit or in a block data program unit.

9.1 DATA Statements

A DATA statement (R9-1) is used to provide initial values for variables, including whole arrays and array elements. A data statement is nonexecutable and may appear anywhere after the IMPLICIT statements, if any, of a program unit.

A *data-stmt* is: R9-1
DATA *data-stmt-set [[,] data-stmt-set]*...

A *data-stmt-set* is: R9-2
data-stmt-object(s) / *data-stmt-value(s)* /

A *data-stmt-object* is one of: R9-3
- scalar-*name*
- array-*name*
- *array-element-designator*
- *data-implied-do*

♦ The *data-stmt-object* must not be a constant.

A *data-stmt-value* is: R9-4
*[repeat-factor *] constant*

A *repeat-factor* is: R9-5
scalar-integer-*constant*

Copyright © 1989–1994 Thinking Machines Corporation

- The *repeat-factor* must be positive. If the repeat factor is a named constant, it must have been declared previously in the program unit containing the DATA statement.
- A name appearing in a *data-stmt-object(s)* list must not be the name of a dummy argument, a procedure, an automatic array, a zero-sized array, or an object in a blank common block. The object list may include the name of an object declared in a named common block provided that the DATA statement is in a block data program unit (Section 9.2).

The *data-stmt-object(s)* list is expanded to form a sequence of scalar variables. An array whose unqualified name appears in a *data-stmt-object(s)* list is equivalent to a complete sequence of its elements in array element order. A *data-implied-do* is expanded to form a sequence of array elements, under the control of the implied DO index variables, as in the loop construct.

The *data-stmt-value(s)* list is expanded to form a sequence of constant values. Each value must be a constant that is previously defined. A repeat factor indicates the number of times the following constant is to be included in the sequence; the constant is included only once if the repeat factor is omitted.

The expanded sequences of scalar variables and constant values are in one-to-one correspondence. Each constant defines the initial value for the corresponding variable. The lengths of the two expanded sequences must be the same.

If a variable is of type character or logical, the corresponding constant must be of the same type. When the variable is of numeric type, the corresponding constant must also be of numeric type. The value of the constant must be compatible with its corresponding variable, according to the rules of assignment (Section 10.1.1), and the constant defines the initial value of the variable according to those rules.

An implicitly typed variable that appears in a DATA statement may appear in a subsequent type declaration only if that declaration confirms the implicit typing.

If a named variable or part of a named variable is initialized in a DATA statement, the named variable has the SAVE attribute, and this may be reaffirmed by a SAVE statement or a type declaration statement containing the SAVE attribute.

A variable, or part of a variable, must not be initialized more than once in a program.

Examples of DATA statements:

```
CHARACTER*12 NAME
INTEGER MILES(10), MARK(3)
DATA NAME /'JOHN DOE'/, MILES /10*0/
DATA MARK(1) /53/, MARK(2) /54/, MARK(3) /95/
```


Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

In the example above, the character variable NAME is initialized with the value JOHN DOE and padded with blanks on the right since the length of the constant is less than the length of the variable. All ten elements of the integer array MILES are initialized to zero. The three elements of the array MARK are initialized individually.

9.1.1 DATA Statements with Implied DO Lists

The use of DATA statements to initialize arrays can be simplified through the use of implied DO control lists, to provide an effect similar to using a loop construct for initialization.

A *data-implied-do* is: R9-6
(*data-implied-do-object(s)*, *data-implied-do-control*)

A *data-implied-do-object* is one of: R9-7
- *array-element-designator*
- *data-implied-do*

A *data-implied-do-control* is: R9-8
index-variable-name = *subscript* , *subscript* [, *stride*]

An *index-variable-name* is: R9-9
scalar-integer-variable-*name*

- The name of a variable appearing in a *data-implied-do-object(s)* list must not be the name of a dummy argument, a procedure, an automatic array, a zero-sized array, or an object in a blank common block. The object list may include the name of an object declared in a named common block provided that the DATA statement is in a block data program unit (Section 9.2).

- The *index-variable-name* and *subscript* and *stride* expressions must conform to the rules of the loop construct, with the additional constraint that the *index-variable-name* in each *data-implied-do-object* must be of type integer.

- A *subscript* or *stride* expression of a *data-implied-do* must involve as primaries only constants or index variable names containing *data-implied-do-control*s.

- A *subscript* in an *array-element-designator* that forms a *data-implied-do-object* must be an expression whose primaries are either constants or index variable names of the containing *data-implied-do-control*s.

- The *index-variable-name* must be a named variable (not an array element designator, an array section, or a substring designator).

Copyright © 1989–1994 Thinking Machines Corporation

As indicated in the previous section, a *data-implied-do* is expanded to form a sequence of array elements, under the control of the implied DO index variables, as in the loop construct.

The following two data statements use implied DO loops to initialize an array with shape [100,100]:

```
DATA ( ( SKEW(I,J), J = 1,I ), I = 1,100 ) / 5050 * 0. /
DATA ( ( SKEW(I,J), J = I+1,100 ), I = 1,99 ) / 4950 * 1. /
```

The two-dimensional array SKEW is initialized so that the lower triangle of SKEW is zero and the strict upper triangle is one.

9.2 Block Data Program Units

A block data program unit (R9-10) specifies initial values for objects in named common blocks. A block data program unit may provide initial values for objects in more than one named common block.

A *block-data-program-unit* is: R9-10

BLOCK DATA *[* block-data-*name]*
[specification-part]
END *[* BLOCK DATA *[* block-data-*name]]*

- The block-data-*name* may be included in the END BLOCK DATA statement only if it was provided in the BLOCK DATA statement; if included, the name must be identical to the block-data-*name* in the BLOCK DATA statement.

- The *specification-part* of a BLOCK DATA statement may contain only IMPLICIT, PARAMETER, INTEGER*4, INTEGER*8, REAL, DOUBLE PRECISION, COMPLEX, DOUBLE COMPLEX, CHARACTER, LOGICAL, COMMON, DIMENSION, EQUIVALENCE, DATA, and SAVE statements.

Only objects in *named* common blocks may be initially defined in a block data program unit. Note that objects associated with an object in a common block (through equivalence association, for example) are considered to be in that common block. If any object in a named common block is initially defined, all objects occupying storage units in the common block storage sequence must be specified in a COMMON statement even if they are not all provided with an initial value.

There must not be more than one unnamed block data program unit in a program.

Version 2.2, October 1994
Copyright © 1989–1994 Thinking Machines Corporation

For CM array data, the common objects themselves must be identically declared across all common blocks in the program.

The same named common block must not be specified in more than one block data program unit in an executable program. The name of a block data program unit has no explicit use in CM Fortran.

When a block data statement is used to initialize arrays in a common block, the `COMMON` directive `$CMF COMMON INITIALIZED` must be used. For more information, see Section A.3.

An example of a block data program unit:

```
BLOCK DATA WORK
  PARAMETER ( VAL = -10. )
  COMMON /WRKCOM/ A, B, C(10,10), SWITCH
  LOGICAL SWITCH
  DATA A /1./, B /2./, C /100*VAL/, SWITCH /.FALSE./
END BLOCK DATA WORK
```

The `BLOCK DATA` program unit above provides an initial value for all variables in the common block `WRKCOM`. All elements of the two-dimensional array C are set initially to the value of the named constant `VAL`. (The constant `VAL` is visible only within the block data program unit.)

9.2.1 Order of Statements in a Block Data Program Unit

The syntax rules given in Section 9.2 and other parts of this document specifying the order of statements within a block data program unit are illustrated in Figure 4. In the figure, vertical lines delineate kinds of statements that may be interspersed and horizontal lines delineate kinds of statements that must not be interspersed.

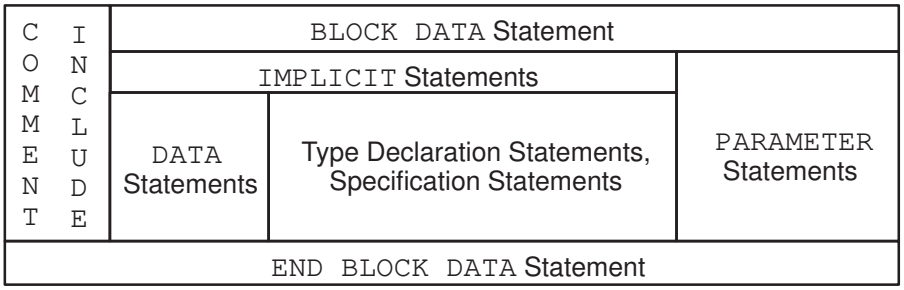

Figure 4. Required order of statements in a block data program unit.

Copyright © 1989–1994 Thinking Machines Corporation

A similar figure is given in Section 3.5.1 for main programs and subprograms, and in Section 12.6.1.1 for interface blocks.

Copyright © 1989–1994 Thinking Machines Corporation