Chapter 16

Scope of Names and Entities

Each name has a *scope*, which is either an executable program, a program unit, an interface block, a single statement, or a part of a statement. Within its scope, a name has a single interpretation. A *scoping unit* is a program unit or an interface block.

- An entity identified by a name whose scope is an executable program is called a *global entity.* The name of a common block is an example of a global entity.
- An entity identified by a name whose scope is a program unit or interface block is called a *local entity.* A named constant defined in a subprogram is an example of a local entity.
- An entity identified by a name whose scope is a single statement or part of a statement is called a *statement entity.* The dummy argument names in a statement function statement are statement entities. The index variable names in an *io-implied-do-control* (R13-13) are statement entities whose scope is part of a statement, namely the *io-implied-do* (R13-12).

By means of association, a single entity may be known by (1) several different names in the same program unit, (2) by the same name in two or more different program units, or (3) by different names in two or more program units.

16.1 Scoping Units and Scope

A program unit consists of one or more disjoint but possibly nested scoping units. A *scoping unit* is:

- A program unit, excluding procedure interface blocks contained within it, or

Copyright © 1989–1994 Thinking Machines Corporation

- A procedure interface block, excluding any procedure interface blocks contained within it.

Note that interface blocks — and hence scoping units — can be nested to a depth of two. In CM Fortran, nested interface blocks may be used for specifying dummy argument names or result types of procedures that are themselves dummy arguments. An example of this is given in Section 12.6.1.

16.2 Scope of Names

The names of variables, constants, statement functions, dummy procedures, intrinsic functions, if constructs, loop constructs, and case constructs have a scope of a program unit or interface block.

The name of a variable that appears as the index variable of an implied `DO` in a `DATA` statement has a scope of the *io-implied-do* (R13-12). It has the type that it would have if it were the name of a variable in the scoping unit that includes the `DATA` statement and this type must be integer.

The name of a statement entity may also be the name of a global or local entity in the same scoping unit; in this case, the name is interpreted within its statement scope as that of the statement entity.

The names of program units, external procedures, and common blocks have a scope of an executable program.

A name that identifies a global entity must not be used to identify any other global entity in the same executable program.

Within a scoping unit, names of entities in the following classes are local entities of that scoping unit.

(1) Program units, common blocks, named constructs, statement functions, dummy procedures, intrinsic functions.

(2) Argument keywords, in a separate class for each procedure with an explicit interface.

A name that identifies a global entity in a scoping unit must not be used to identify a local entity of class (1) in that scoping unit.

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

Within a scoping unit, a name that identifies a local entity of one class must not be used to identify another entity of the same class. A name that identifies a local entity of one class may be used to identify a local entity of another class.

The name of a local entity identifies that entity in a scoping unit and may also be used to identify any local or global entity in another scoping unit.

16.2.1 Common Block Names

A common block name in a program unit may be the name of any local entity of that program unit other than (1) a named constant, (2) an intrinsic function, or (3) a local variable, if the name of that variable is also the name of an external function referenced within the program unit (whether it is declared implicitly or explicitly). If a name *is* used for both a common block and a local entity, the appearance of that name in any context other than as a common block name in a `COMMON` or `SAVE` statement identifies the local entity. Note that a common block name may be the name of an intrinsic function in a program unit that does not reference the intrinsic function.

16.2.2 Function Result Variables

A function subprogram must declare (either explicitly or implicitly) a local (result) variable with the same name as the function. If a function subprogram contains one or more `ENTRY` statements, there must be a local variable with the same name as each entry.

16.2.3 Argument Keywords

A dummy argument name in a procedure interface block has a scope as an argument keyword of the scoping unit of the containing program unit. As an argument keyword, it may appear only in a procedure reference for a procedure of which it is a dummy argument name.

16.2.4 Statement Entities

The name of a variable that appears as a dummy argument in a statement function statement has a scope of the statement in which it appears. It has the type that it would have if it were the name of a variable in the program unit that declares the statement function.

Copyright © 1989–1994 Thinking Machines Corporation

The name of a variable that appears as the index variable of an implied DO in a DATA statement has a scope of the implied DO (R13-13).

The name of a statement entity may also be the name of a global or local entity in the same program unit or interface block; in this case, the name is interpreted within its statement scope as that of the statement variable.

16.3 Scope of Other Entities

A label has a scope of a program unit. No two statements in the same program unit may have the same label.

An external input/output unit has the scope of an executable program.

The operators have a scope of an executable program.

16.4 Events That Change the Value of Variables

The value of a variable becomes defined as a result of any of the following actions:

- Execution of an assignment statement other than a masked array assignment causes the variable on the left-hand side of the assignment to become defined.
- Execution of a masked array assignment statement may cause some or all of the array elements in the assignment statement to become defined.
- Execution of an input statement causes each variable in its input list to become defined with a value from the input file when the data is transferred to it (assuming no input/output errors).
- Execution of a DO statement causes the index variable, if any, to become defined.
- Beginning of execution of the action specified by an implied DO in an input/output statement causes the index variable to become defined.
- Execution of an ASSIGN statement causes the variable in the statement to become defined with a statement label value.

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

- A reference to a procedure causes a component of a dummy argument to become defined if the corresponding component of the actual argument is assigned a value that is not a statement label.
- Execution of an input/output statement containing an `IOSTAT=` specifier causes the specified integer variable to become defined.
- Execution of an input/output statement containing a `NULL=` specifier causes the specified integer variable to become defined.
- Execution of an `INQUIRE` statement causes any variable that is assigned a value during the execution of the statement to become defined if no error condition results.
- An action that causes a character storage unit to become defined also causes all associated character storage units to become defined. An action that causes a numeric storage unit to become defined also causes all associated numeric storage units of the same type become defined, except that variables associated with the variable in an `ASSIGN` statement become undefined when the `ASSIGN` statement is executed.
- An action that causes a scalar variable without a storage sequence to become defined also causes all associated variables to become defined.
- Invocation of a procedure causes any automatic object of size zero to become defined.

The value of a variable becomes undefined as a result of any of the following actions:

- An action that causes a variable of a given type to become defined also causes all associated variables of different type to become undefined. However, when a variable of type real is partially associated with a variable of type complex (both variables of the same precision), the complex variable does not become undefined when the real variable becomes defined, and vice versa. When a variable of type complex is partially associated with another variable of type complex (both of the same precision), definition of one does not cause the other to become undefined.
- Execution of an `ASSIGN` statement causes the variable in the statement to become undefined as an integer. Variables that are associated with the variable also become undefined.
- If the evaluation of a function may cause an argument of the function or a variable in a common block to become undefined, and if a reference to the function appears in an expression in which the value of the function is not needed to determine the

Copyright © 1989–1994 Thinking Machines Corporation

value of the expression, the argument or variable becomes undefined when the expression is evaluated.

- The execution of a RETURN statement or END statement within a subprogram causes all variables local to it to become undefined except for the following:
  - Variables with the SAVE attribute.
  - Variables in blank common.
  - Variables in a named common block that appear in the subprogram and also appear in at least one other program unit that makes either a direct or an indirect reference to the subprogram.
  - Initially defined variables whose value has not changed.
- When an error condition or end-of-file condition occurs during execution of an input statement, all the variables specified by the input list of the statement become undefined.
- When an error condition or end-of-file condition occurs during execution of an input/output statement, some or all of the implied DO index variables may become undefined.
- Execution of a direct access input statement that specifies a record that has not been written previously causes all the variables specified by the input list of the statement to become undefined.
- Execution of an INQUIRE statement may cause the variables referenced by the NAME=, RECL=, and NEXTREC= specifiers to become undefined.
- When a character storage unit becomes undefined, all associated character storage units become undefined.

  When a numeric storage unit becomes undefined, all associated numeric storage units become undefined unless it becomes undefined as a result of defining an associated numeric storage unit of different type. (See the first item, above, describing the effects of individually assigning to the real or imaginary part of a complex number.)

  When a scalar variable without a storage sequence becomes undefined, all associated variables become undefined.
- A reference to a procedure causes part of a dummy argument to become undefined if the corresponding part of the actual argument is defined with a value that is a statement label.

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

- Execution of an `INQUIRE` statement causes all inquiry specifier variables to become undefined if an error condition exists, except for the variable in the `IOSTAT=` specifier, if any.
- When a procedure is invoked, a component object of one of its dummy arguments is undefined if the corresponding component object of the actual argument is undefined. If a procedure has a dummy argument with attribute `INTENT(OUT)`, then the actual argument corresponding to that dummy argument is undefined when the procedure is invoked.

16.5 Use of Whole Array Names

Table 25 indicates every context in which a whole array name can occur.

Table 25. Contexts in which whole array names can appear.

Place of Appearance	Explicit Shape	Assumed-Shape	Assumed-Size
dummy-arg (R12-2)	yes	yes	yes
type-declaration-stmt (R5-10)	yes	yes	yes
equivalence-stmt [1] (R8-4)	yes	no	no
data-stmt (R9-1)	yes	no	no
common-stmt (R8-1)	yes	no	no
input-item(s) or *output-item(s)* (R13-9)	yes	yes	no
internal-file-spec (R13-7)	yes	no	no
format-specification (R15-1)	yes	no	no
forall-stmt [2] (R10-7)	yes	yes	yes
save-stmt (R5-55)	yes	no	no
primary (R7-2)	yes	yes	no
assignment-stmt (R11-30)	yes	yes	no
actual-arg (R12-9)	yes	yes	yes
where-stmt (R10-2)	yes	yes	no

[1] Only front-end arrays may be used in `EQUIVALENCE` statements.

[2] On the right-hand-side only.

Copyright © 1989–1994 Thinking Machines Corporation