About This Manual

Objectives

This manual defines the CM Fortran language, including the compiler directives.

Intended Audience

Readers of this manual will find previous programming experience with Fortran 77 helpful but not essential. Reference to either the Fortran 77 standard or the Fortran 90 standard is not required.

Organization of This Manual

Chapter 1 introduces the language CM Fortran and the data parallel execution model. Chapters 2 through 17 define the language, with Chapter 17 defining the intrinsic functions provided by CM Fortran. These include all the intrinsic functions of Fortran 77 and many of the intrinsic functions of Fortran 90 that deal with arrays. Chapter 18 covers some of the implementation-dependent aspects of array homes and argument passing in CM Fortran. Appendix A describes compiler directives and their effect on the execution of programs.

Revision Information

This version supersedes the *CM Fortran Reference Manual* Version 2.1, January 1994.

Copyright © 1989–1994 Thinking Machines Corporation

Related Documents

- *Getting Started in CM Fortran*, October 1994
- *CM Fortran User's Guide*, Version 2.2
- *CM Fortran Libraries Reference Manual*, Version 2.2
- *CM Fortran Programming Guide*, Version 2.2
- *CM-5 CM Fortran Performance Guide*, Version 2.2

Notation Conventions for Syntax Rules

Syntax rules are used to describe the valid forms that lexical tokens, statements, and constructs may take. Syntax rule names appear in an italic sans serif typeface, as in *program-unit* or *executable-stmt*. These syntax rules are expressed using a variation of Backus-Naur form (BNF) in which:

- Roman characters from the CM Fortran character set that form part of a syntactic construct are to be written as they appear, except where noted otherwise. For example,

  ```
  PROGRAM
  ```

  represents the seven characters specified, in that order.

- Lowercase *italic* letters and words (often hyphenated and abbreviated) represent general syntactic classes for which specific syntactic entities must be substituted in actual statements or constructs.

- A syntactic class can be defined as a sequence of classes or constructs using a *rule definition* of the form

  A *category* is a: R*m*-*n*
  item1 item2
  item3
  item4

  where *category* is a symbolic name for the class, and the *item*s can be characters from the character set, syntactic constructs, or the names of other syntactic classes (italicized). One or more *item*s may be placed on a line within the definition. Items are usually placed on the same line if they are required to appear on the same line in the source text (but see the last rule below). Each syntactic class is labeled on the right with a rule number of the form R*m*-*n*, where *m* is the number of the chapter in which the rule is defined and *n* is its serial number within that chapter.

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

A more specialized view of a particular syntactic category may be introduced using a qualifying phrase before the category name, as:

A (particular kind of) *category* is a:
...

where the parenthesized phrase "particular kind of" names a specific form of a more general *category* defined in another class definition. The syntax of such specialized definitions is restricted in some way compared with the syntax of the more general rules, to reflect the specialization.

For convenience, the same syntactic rule may be repeated in several places throughout the manual. All duplicate occurrences of a rule are given the original rule number.

- A syntactic class can be defined as a choice among several alternative classes or constructs using:

  A *selection* is one of:
  - *item1*
  - *item2a item2b*
  - *item3*

- Each alternative list of *item*s is introduced by a small bullet (•).

- Within a syntactic class definition, items enclosed in italic brackets are optional:

  [optional-item another-optional-item]

- Within a syntactic class definition, items enclosed in italic brackets immediately followed by an ellipsis may be repeated zero or more times:

  [optional-repeated-item a-second-item]...

Syntactic constraints imposed on a rule by surrounding context are specified in English sentences introduced by a leading diamond symbol (♦), as in:

A *foo-stmt* is:
`FOO` (foo-arg-*name*)

♦ A foo-arg-*name* must not have been used previously in a `BAR` statement.

The constraint imposes a restriction on the kinds of *name* that can be used in a `FOO` statement.

The rule above defining *foo-stmt* also illustrates a naming convention: only the italicized portion of a name has a corresponding definition in the syntax, and an unitalicized prefix

Copyright © 1989–1994 Thinking Machines Corporation

in a name imposes some context constraint that should be apparent. The example above references the syntactic name foo-arg-*name*, but only *name* is defined in a syntax rule.

To reduce the number of syntax rules, a symbolic name ending in "*(s)*" is assumed to be defined as a list of one or more items with the specified name, separated by commas. For example, a reference to *xyz(s)* or to "an *xyz(s)* list" implicitly defines the following rule:

An *xyz(s)* is:
xyz [, xyz]...

The symbol *xyz* should be defined explicitly by a syntax rule elsewhere in the document.

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