Chapter 2

Language Elements

This chapter describes the CM Fortran character set and the form of low-level language elements such as names and operators.

2.1 Character Set

The Fortran character set consists of 26 letters, 10 digits, underscore, dollar sign, and 19 special characters.

A *character* is one of: R2-1
- *alphanumeric-character*
- *underscore*
- *dollar-sign*
- *special-character*

An *alphanumeric-character* is one of: R2-2
- *letter*
- *digit*

A *letter* is: R2-3
one of the 26 characters of the alphabet. Upper- and lowercase letters are equivalent except in character constants and delimited character edit descriptors.

A *digit* is: R2-4
one of the 10 decimal digits

An *underscore* is: R2-5
_

A *dollar-sign* is: R2-6
$

Copyright © 1989–1994 Thinking Machines Corporation

A *special-character* is one of the following 19 characters (*sp* represents the space character):

```
sp !  "  &  '  (  )
*  +  ,  -  .  /  :
<  =  >  [  ]
```

2.2 Lexical Syntax

The lexical syntax describes the form of the fundamental text units of a CM Fortran program. These *lexical tokens* are sequences of characters and include keywords, names, delimiters, literal constants, operator symbols, and statement labels.

Keywords, names, delimiters, and labels are described in this chapter. Literal constants are described in Chapter 4. Operator symbols are formed by sequences of special characters, or are sequences of letters surrounded by delimiter characters. Operator symbols are described in Section 7.2.

2.2.1 Keywords

A sequence of characters that is part of the syntax of a statement and that may be used to identify the statement is a *statement keyword.* Examples of keywords are: `IF`, `READ`, `WHERE`, and `INTEGER`. These keywords are not "reserved words" — names with the same spellings may be defined in a program unit. Keywords appear as uppercase words in syntax rules of this manual.

2.2.2 Names

A *name* (R2-7) is a sequence of characters beginning with a letter or an underscore, followed by letters, digits, and underscores. A name identifies a program component, such as a program unit, named variable, named constant, or dummy argument. Names may contain uppercase and lowercase letters interchangeably (case is not significant), digits, and underscores.

A *name* is one of: R2-7

- *letter [alphanumeric-or-other-character]...*
- *underscore [alphanumeric-or-other-character]...* ! (see **NOTE**)

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

An *alphanumeric-or-other-character* is one of: R2-8

- *alphanumeric-character*
- *underscore*
- *dollar-sign*

- The maximum length of a name is 31 characters.
- A name cannot be used as a global symbol and and a local symbol in the same program unit. For example, a program-*name* (see Section 3.5) is global to the program, and must not be the same as the name of any variable, which is a local symbol.

NOTE: Unlike standard Fortran 77 and Fortran 90, CM Fortran allows the underscore character to be used as a leading character in identifiers. This enables CM Fortran subprograms to call external subprograms written in C, which have an initial underscore prefixed by the UNIX C compiler. Also, CM Fortran allows the $ character to be used in names, but not as the first character.

No implicit type is assumed for a name beginning with an underscore; the type of such a name must be specified explicitly using a type declaration statement.

Examples of valid names:

```
A1
NAME_LENGTH
T_E_S_T
_C_ROUTINE  ! no implicit type
alpha_1
beta2
_           ! no implicit type
A$FOO
```

2.2.3 Delimiters

A *delimiter* is one of a pair of characters or character sequences used to enclose character constants, array constructors, operator symbols, and other lexical tokens. The following are the delimiters used in CM Fortran, along with a note regarding their use.

```
(     )     ! expressions (Chapter 7)
/     /     ! DATA statements (Section 9.1)
"     "     ! character strings (Section 4.3)
'     '     !     "    "
.     .     ! operator symbols (Section 7.2)
(/    /)    ! array constructors (Section 7.1.7)
[     ]     !   "    "
```


Copyright © 1989–1994 Thinking Machines Corporation

2.3 Lines

A *line* in a program unit is normally a sequence of 72 characters. The maximum length of a line may be increased to 132 characters by using the `-extend_source` compiler switch, described in the *CM Fortran User's Guide.* A statement is written on one or more lines, the first of which is called an *initial line*; succeeding lines, if any, are called *continuation* lines. There may also be one or more *comment lines* that are not part of any statement, but are intended to provide documentation.

Lines beginning with a `C` or `*` in column position 1 and lines containing only blanks are comment lines. Comment lines may appear anywhere within a program unit and may precede the first statement of a program unit. The character `!` initiates a comment except when it appears in a character context or in column 6. The comment extends to the end of the line. If the first nonblank character on a line is an `!` in any column other than column 6, the line is also a comment line. CM Fortran does not limit the number of comment lines in a program unit.

Comment lines and comments normally have no effect on the interpretation of a program. CM Fortran recognizes several *directives* in the form of comment lines with a special syntax; such directives may have an effect on the interpretation of a program. Directives are described in Appendix A.

An initial line is a line that contains a blank in column 6 (and is not a comment line).

A continuation line is any line containing a character other than 0 (zero) or blank in column 6. An initial line may be followed by up to 19 continuation lines, but this number may be increased by using the `-continuations` compiler switch, described in the *CM Fortran User's Guide.*

NOTE: CM Fortran regards a tab character in column one of a line as equivalent to 6 blank characters.

Examples of lines:

```
C234567890
C     This is a comment line
*     ...so is this
!     ...and this

C An assignment statement extending over three lines:
      NAME =               ! an initial line
     x   "!alpha!beta" //  ! a continuation line
     !         "!gamma"    ! another continuation line
```


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

All but the last three lines above are comment lines. The value assigned to NAME in the example is the string '!alpha!beta!gamma'.

2.3.1 INCLUDE Line

The INCLUDE line (R2-9) causes text from a file to be considered part of the text of a program unit.

An *include-line* is: R2-9
INCLUDE *file-specification*

The *file-specification* is a character literal constant (R4-19, p. 29) specifying the name of the file to be included. An INCLUDE line of the form

```
INCLUDE "filename"
```

effectively causes the contents of file *filename* to replace the INCLUDE line.

Files to be included can be specified with either absolute or relative pathnames, or can be found using the -I switch. For more information, see the *CM Fortran User's Guide*.

The INCLUDE line may appear anywhere within a source file that a statement can appear. The INCLUDE line may not be continued over more than one line. A file to be included must contain only complete statements or comment lines or other INCLUDE lines. That is, no single statement may be taken from more than one source file.

2.4 Statements and Constructs

A *statement* describes or controls a computational action, or prescribes the program environment in which computational actions take place, as described in Section 3.1. A statement is formed from combinations of keywords, expressions, constants, variable names, and subprogram names. Statements may be written on a single line (in columns 7 through 72), or may extend over several lines using the conventions for continuation lines described above. Some statements appear only as part of other statements, and the whole collection of related statements taken together is called a *construct*.

IF constructs (Section 11.3), CASE constructs (Section 11.4), and block loop constructs (Section 11.5.1) can be identified with a *construct name* by preceding the first statement of the construct with a name, separated from the statement by a colon (:).

Copyright © 1989–1994 Thinking Machines Corporation

Within this manual, any syntax class name ending in *"-stmt"* identifies a statement. A syntax class name ending in "*-construct"* identifies a construct.

An example of an IF construct:

```
IF ( CVAR .EQ. "RESET" ) THEN
  I = 0
  J = 0
  K = 0
END IF
```

The IF–THEN statement is always paired with the END IF statement. As the example illustrates, it may be helpful to indent nested statements or constructs with respect to the containing construct to indicate the containment relationship.

An example of a DO construct with a construct name OUTER:

```
OUTER: DO I = 1,10
  DO J = 1,5
    ...
    EXIT OUTER
    ...
  END DO
END DO OUTER
```

2.4.1 Statement Labels

A statement *label* (R2-10) provides a means of referring to an individual statement.

A *label* is: R2-10
digit [digit [digit [digit [digit]]]]

Statement labels may be placed in columns 1 through 5 of the initial line of a statement. Blanks and leading zeros are not significant in distinguishing statement labels. Blanks may appear within a label. If a line is labeled, it must contain at least one nonblank character of a statement. The same statement label must not be given to more than one statement in a program unit.

Any statement that begins a line may be identified with a label, but not all statements may be referred to by the use of labels. Branch target statements may be referred to by branching statements (Section 11.6) and the ASSIGN statement (Section 11.6.3), FORMAT statements (Section 15.1) may be referred to by input/output statements (Chapter 13) and the ASSIGN statement, and loop termination statements may be referred to by loop constructs (Section 11.5) and branching statements.

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

Note that a statement label and a construct name (Section 2.4) may appear on the same line.

Following are examples of statement labels on `CONTINUE` statements:

```
10     CONTINUE
 1 0   CONTINUE
010    CONTINUE
9999   CONTINUE
```

The first three labels are equivalent.

Copyright © 1989–1994 Thinking Machines Corporation