Section 3: Characters, lexical tokens, and source form

This section describes the Fortran character set and the various lexical tokens such as names and operators. This section also describes the rules for the forms that Fortran programs may take.

3.1 Processor character set

The processor character set is processor dependent. The structure of a processor character set is:

(1) **Control characters** ("newline", for example)

(2) **Graphic characters**

- (a) Letters (3.1.1)
- (b) Digits (3.1.2)
- (c) Underscore (3.1.3)
- (d) Special characters (3.1.4)
- (e) Other characters (3.1.5)

The letters, digits, underscore, and special characters make up the **Fortran character set**.

R301	*character*	**is**	*alphanumeric-character*
		or	*special-character*
R302	*alphanumeric-character*	**is**	*letter*
		or	*digit*
		or	*underscore*

Except for the currency symbol, the graphics used for the characters shall be as given in 3.1.1, 3.1.2, 3.1.3, and 3.1.4. However, the style of any graphic is not specified.

3.1.1 Letters

The twenty-six **letters** are:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The set of letters defines the syntactic class *letter*.

If a processor also permits lower-case letters, the lower-case letters are equivalent to the corresponding upper-case letters in program units except in a character context (3.3).

3.1.2 Digits

The ten **digits** are:

0 1 2 3 4 5 6 7 8 9

The ten digits define the syntactic class *digit*.

3.1.3 Underscore

R303	*underscore*	**is**	_

The underscore may be used as a significant character in a name.

3.1.4 Special characters

The twenty-one **special characters** are shown in Table 3.1.

Table 3.1 Special characters

Character	Name of character	Character	Name of character
	Blank	:	Colon
=	Equals	!	Exclamation point
+	Plus	"	Quotation mark or quote
-	Minus	%	Percent
*	Asterisk	&	Ampersand
/	Slash	;	Semicolon
(	Left parenthesis	<	Less than
)	Right parenthesis	>	Greater than
,	Comma	?	Question mark
.	Decimal point or period	$	Currency symbol
'	Apostrophe		

The twenty-one special characters define the syntactic class *special-character*. The special characters are used for operator symbols, bracketing, and various forms of separating and delimiting other lexical tokens. The special characters $ and ? have no specified use.

3.1.5 Other characters

Additional characters may be representable in the processor, but may appear only in comments (3.3.1.1, 3.3.2.1), character constants (4.3.2.1), input/output records (9.1.1), and character string edit descriptors (10.2.1).

The default character type shall support a character set that includes the Fortran character set. Other character sets may be supported by the processor in terms of nondefault character types. The characters available in the nondefault character types are not specified, except that one character in each nondefault character type shall be designated as a blank character to be used as a padding character.

3.2 Low-level syntax

The **low-level syntax** describes the fundamental lexical tokens of a program unit. **Lexical tokens** are sequences of characters that constitute the building blocks of a program. They are keywords, names, literal constants other than complex literal constants, operators, labels, delimiters, comma, =, =>, :, ::, ;, and %.

3.2.1 Names

Names are used for various entities such as variables, program units, dummy arguments, named constants, and derived types.

R304 *name* **is** *letter* [*alphanumeric-character*] ...

Constraint: The maximum length of a *name* is 31 characters.

NOTE 3.1

Examples of names:

```
A1
NAME_LENGTH              (single underscore)
S_P_R_E_A_D__O_U_T       (two consecutive underscores)
TRAILER_                 (trailing underscore)
```

3.2.2 Constants

R305	*constant*	**is**	*literal-constant*
		or	*named-constant*
R306	*literal-constant*	**is**	*int-literal-constant*
		or	*real-literal-constant*
		or	*complex-literal-constant*
		or	*logical-literal-constant*
		or	*char-literal-constant*
		or	*boz-literal-constant*
R307	*named-constant*	**is**	*name*
R308	*int-constant*	**is**	*constant*

Constraint: *int-constant* shall be of type integer.

R309	*char-constant*	**is**	*constant*

Constraint: *char-constant* shall be of type character.

3.2.3 Operators

R310	*intrinsic-operator*	**is**	*power-op*
		or	*mult-op*
		or	*add-op*
		or	*concat-op*
		or	*rel-op*
		or	*not-op*
		or	*and-op*
		or	*or-op*
		or	*equiv-op*
R708	*power-op*	**is**	**
R709	*mult-op*	**is**	*
		or	/
R710	*add-op*	**is**	+
		or	–
R712	*concat-op*	**is**	//
R714	*rel-op*	**is**	.EQ.
		or	.NE.
		or	.LT.
		or	.LE.
		or	.GT.
		or	.GE.
		or	==
		or	/=
		or	<

or <=
or >
or >=

R719 *not-op* **is** .NOT.

R720 *and-op* **is** .AND.

R721 *or-op* **is** .OR.

R722 *equiv-op* **is** .EQV.
or .NEQV.

R311 *defined-operator* **is** *defined-unary-op*
or *defined-binary-op*
or *extended-intrinsic-op*

R704 *defined-unary-op* **is** . *letter* [*letter*] ... .

R724 *defined-binary-op* **is** . *letter* [*letter*] ... .

R312 *extended-intrinsic-op* **is** *intrinsic-operator*

Constraint: A *defined-unary-op* and a *defined-binary-op* shall not contain more than 31 letters and shall not be the same as any *intrinsic-operator* or *logical-literal-constant*.

3.2.4 Statement labels

A statement label provides a means of referring to an individual statement.

R313 *label* **is** *digit* [*digit* [*digit* [*digit* [*digit*]]]]

Constraint: At least one digit in a *label* shall be nonzero.

If a statement is labeled, the statement shall contain a nonblank character. The same statement label shall not be given to more than one statement in a scoping unit. Leading zeros are not significant in distinguishing between statement labels.

NOTE 3.2

For example:

```
99999
10
 010
```

are all statement labels. The last two are equivalent.

There are 99999 unique statement labels and a processor shall accept any of them as a statement label. However, a processor may have an implementation limit on the total number of unique statement labels in one program unit.

3.2.5 Delimiters

Delimiters are used to enclose syntactic lists. The following pairs are delimiters:

```
( ... )
/ ... /
(/ ... /)
```

3.3 Source form

A Fortran program unit is a sequence of one or more lines, organized as Fortran statements, comments, and INCLUDE lines. A **line** is a sequence of zero or more characters. Lines following a program unit END statement are not part of that program unit. A Fortran **statement** is a sequence of one or more complete or partial lines.

A **character context** means characters within a character literal constant (4.3.2.1) or within a character string edit descriptor (10.2.1).

A comment may contain any character that may occur in any character context.

There are two **source forms**: free and fixed. Free form and fixed form shall not be mixed in the same program unit. The means for specifying the source form of a program unit are processor dependent.

3.3.1 Free source form

In **free source form**, each source line may contain from zero to 132 characters and there are no restrictions on where a statement (or portion of a statement) may appear within a line. However, if a line contains any character that is not of default kind (4.3.2.1), the number of characters allowed on the line is processor dependent.

Blank characters shall not appear within lexical tokens other than in a character context or in a format specification. Blanks may be inserted freely between tokens to improve readability; for example, blanks may occur between the tokens that form a complex literal constant. A sequence of blank characters outside of a character context is equivalent to a single blank character.

A blank shall be used to separate names, constants, or labels from adjacent keywords, names, constants, or labels.

NOTE 3.3

For example, the blanks after REAL, READ, 30, and DO are required in the following:

```
REAL X
READ 10
30 DO K=1,3
```

One or more blanks shall be used to separate adjacent keywords except in the following cases, where blanks are optional:

Adjacent keywords where separating blanks are optional
BLOCK DATA
DOUBLE PRECISION
ELSE IF
END BLOCK DATA
END DO
END FILE
END FORALL
END FUNCTION
END IF
END INTERFACE
END MODULE
END PROGRAM
END SELECT
END SUBROUTINE
END TYPE
END WHERE
GO TO
IN OUT
SELECT CASE

NOTE 3.4

Allowing optional blanks at specific places in some keywords (for example, ENDIF or END IF) is intended to permit a reasonable choice to users accustomed to insignificant blanks.

In some circumstances, for example where source code is maintained in an INCLUDE file for use in programs whose source form might be either fixed or free, observing the following rules allows the code to be used with either source form:

(1) Confine statement labels to character positions 1 to 5 and statements to character positions 7 to 72;

(2) Treat blanks as being significant;

(3) Use only the exclamation mark (!) to indicate a comment, but do not start the comment in character position 6;

(4) For continued statements, place an ampersand (&) in both character position 73 of a continued line and character position 6 of a continuing line.

3.3.1.1 Free form commentary

The character "!" initiates a **comment** except when it appears within a character context. The comment extends to the end of the source line. If the first nonblank character on a line is an "!", the line is called a comment line. Lines containing only blanks or containing no characters are also comment lines. Comments may appear anywhere in a program unit and may precede the first statement of a program unit. Comments have no effect on the interpretation of the program unit.

NOTE 3.5

The standard does not restrict the number of consecutive comment lines.

3.3.1.2 Free form statement separation

The character ";" terminates a statement, except when the ";" appears in a character context or in a comment. This optional termination allows another statement to begin following the ";" on the same line. A ";" shall not appear as the first nonblank character on a line. If a ";" separator is followed by zero or more blanks and one or more ";" separators, the sequence from the first ";" to the last, inclusive, is interpreted as a single ";" separator.

3.3.1.3 Free form statement continuation

The character "&" is used to indicate that the current statement is continued on the next line that is not a comment line. Comment lines shall not be continued; an "&" in a comment has no effect. Comments may occur within a continued statement. When used for continuation, the "&" is not part of the statement. No line shall contain a single "&" as the only nonblank character or as the only nonblank character before an "!" that initiates a comment.

3.3.1.3.1 Noncharacter context continuation

If an "&" not in a comment is the last nonblank character on a line or the last nonblank character before an "!", the statement is continued on the next line that is not a comment line. If the first nonblank character on the next noncomment line is an "&", the statement continues at the next character position following the "&"; otherwise, it continues with the first character position of the next noncomment line.

If a lexical token is split across the end of a line, the first nonblank character on the first following noncomment line shall be an "&" immediately followed by the successive characters of the split token.

3.3.1.3.2 Character context continuation

If a character context is to be continued, the "&" shall be the last nonblank character on the line and shall not be followed by commentary. An "&" shall be the first nonblank character on the next line that is not a comment line and the statement continues with the next character following the "&".

3.3.1.4 Free form statements

A label may precede any statement not forming part of another statement.

NOTE 3.6

No Fortran statement begins with a digit.

A free form statement shall not have more than 39 continuation lines.

3.3.2 Fixed source form

In **fixed source form**, there are restrictions on where a statement may appear within a line. If a source line contains only default kind characters, it shall contain exactly 72 characters; otherwise, its maximum number of characters is processor dependent.

Except in a character context, blanks are insignificant and may be used freely throughout the program.

3.3.2.1 Fixed form commentary

The character "!" initiates a **comment** except when it appears within a character context or in character position 6. The comment extends to the end of the line. If the first nonblank character on a line is an "!" in any character position other than character position 6, the line is a comment line. Lines beginning with a "C" or "*" in character position 1 and lines containing only blanks are also comments. Comments may appear anywhere within a program unit and may precede the first statement of the program unit. Comments have no effect on the interpretation of the program unit.

NOTE 3.7

The standard does not restrict the number of consecutive comment lines.

3.3.2.2 Fixed form statement separation

The character ";" terminates a statement, except when the ";" appears in a character context, in a comment, or in character position 6. This optional termination allows another statement to begin following the ";" on the same line. A ";" shall not appear as the first nonblank character on a line, except in character position 6. If a ";" separator is followed by zero or more blanks and one or more ";" separators, the sequence from the first ";" to the last, inclusive, is interpreted as a single ";" separator.

3.3.2.3 Fixed form statement continuation

Except within commentary, character position 6 is used to indicate continuation. If character position 6 contains a blank or zero, the line is the initial line of a new statement, which begins in character position 7. If character position 6 contains any character other than blank or zero, character positions 7–72 of the line constitute a continuation of the preceding noncomment line.

NOTE 3.8

An "!" or ";" in character position 6 is interpreted as a continuation indicator unless it appears within commentary indicated by a "C" or "*" in character position 1 or by an "!" in character positions 1–5 (3.3.2.3).

Comment lines shall not be continued. C omment lines may occur within a continued statement.

3.3.2.4 Fixed form statements

A label, if present, shall occur in character positions 1 through 5 of the first line of a statement; otherwise, positions 1 through 5 shall be blank. Blanks may appear anywhere within a label. A statement following a ";" on the same line shall not be labeled. Character positions 1 through 5 of any continuation lines shall be blank. A fixed form statement shall not have more than 19 continuation lines. The program unit END statement shall not be continued. A statement whose initial line appears to be a program unit END statement shall not be continued.

3.4 Including source text

Additional text may be incorporated into the source text of a program unit during processing. This is accomplished with the **INCLUDE line**, which has the form

INCLUDE *char-literal-constant*

The *char-literal-constant* shall not have a kind type parameter value that is a *named-constant*.

An INCLUDE line is not a Fortran statement.

An INCLUDE line shall appear on a single source line where a statement may appear; it shall be the only nonblank text on this line other than an optional trailing comment. Thus, a statement label is not allowed.

The effect of the INCLUDE line is as if the referenced source text physically replaced the INCLUDE line prior to program processing. Included text may contain any source text, including additional INCLUDE lines; such nested INCLUDE lines are similarly replaced with the specified source text. The maximum depth of nesting of any nested INCLUDE lines is processor dependent. Inclusion of the source text referenced by an INCLUDE line shall not, at any level of nesting, result in inclusion of the same source text.

When an INCLUDE line is resolved, the first included statement line shall not be a continuation line and the last included statement line shall not be continued.

The interpretation of *char-literal-constant* is processor dependent. An example of a possible valid interpretation is that *char-literal-constant* is the name of a file that contains the source text to be included.