Information technology — Programming languages — Fortran —

Part 1:
Base language

Section 1: Overview

1.1 Scope

ISO/IEC 1539 is a multipart International Standard; the parts are published separately. This publication, ISO/IEC 1539-1, which is the first part, specifies the form and establishes the interpretation of programs expressed in the base Fortran language. The purpose of this part of ISO/IEC 1539 is to promote portability, reliability, maintainability, and efficient execution of Fortran programs for use on a variety of computing systems. The second part, ISO/IEC 1539-2, defines additional facilities for the manipulation of character strings of variable length. A processor conforming to part 1 need not conform to ISO/IEC 1539-2; however, conformance to ISO/IEC 1539-2 assumes conformance to this part. Throughout this publication, the term "this standard" refers to ISO/IEC 1539-1.

1.2 Processor

The combination of a computing system and the mechanism by which programs are transformed for use on that computing system is called a **processor** in this standard.

1.3 Inclusions

This standard specifies

(1) The forms that a program written in the Fortran language may take,
(2) The rules for interpreting the meaning of a program and its data,
(3) The form of the input data to be processed by such a program, and
(4) The form of the output data resulting from the use of such a program.

1.4 Exclusions

This standard does not specify

(1) The mechanism by which programs are transformed for use on computing systems,
(2) The operations required for setup and control of the use of programs on computing systems,
(3) The method of transcription of programs or their input or output data to or from a storage medium,
(4) The program and processor behavior when this standard fails to establish an interpretation except for the processor detection and reporting requirements in items (2) through (8) of 1.5,
(5) The size or complexity of a program and its data that will exceed the capacity of any specific computing system or the capability of a particular processor,
(6) The physical properties of the representation of quantities and the method of rounding, approximating, or computing numeric values on a particular processor,

(7) The physical properties of input/output records, files, and units, or

(8) The physical properties and implementation of storage.

1.5 Conformance

A program (2.2.1) is a **standard-conforming program** if it uses only those forms and relationships described herein and if the program has an interpretation according to this standard. A program unit (2.2) conforms to this standard if it can be included in a program in a manner that allows the program to be standard conforming.

A processor conforms to this standard if

(1) It executes any standard-conforming program in a manner that fulfills the interpretations herein, subject to any limits that the processor may impose on the size and complexity of the program;

(2) It contains the capability to detect and report the use within a submitted program unit of a form designated herein as obsolescent, insofar as such use can be detected by reference to the numbered syntax rules and their associated constraints;

(3) It contains the capability to detect and report the use within a submitted program unit of an additional form or relationship that is not permitted by the numbered syntax rules or their associated constraints, including the deleted features described in Annex B;

(4) It contains the capability to detect and report the use within a submitted program unit of kind type parameter values (4.3) not supported by the processor;

(5) It contains the capability to detect and report the use within a submitted program unit of source form or characters not permitted by Section 3;

(6) It contains the capability to detect and report the use within a submitted program of name usage not consistent with the scope rules for names, labels, operators, and assignment symbols in Section 14;

(7) It contains the capability to detect and report the use within a submitted program unit of intrinsic procedures whose names are not defined in Section 13; and

(8) It contains the capability to detect and report the reason for rejecting a submitted program.

However, in a format specification that is not part of a FORMAT statement (10.1.1), a processor need not detect or report the use of deleted or obsolescent features, or the use of additional forms or relationships.

A standard-conforming processor may allow additional forms and relationships provided that such additions do not conflict with the standard forms and relationships. However, a standard-conforming processor may allow additional intrinsic procedures even though this could cause a conflict with the name of a procedure in a standard-conforming program. If such a conflict occurs and involves the name of an external procedure, the processor is permitted to use the intrinsic procedur
scoping unit (14). A standard-conforming program shall not use nonstandard intrinsic procedures that have been added by the processor.

Because a standard-conforming program may place demands on a processor that are not within the scope of this standard or may include standard items that are not portable, such as external procedures defined by means other than Fortran, conformance to this standard does not ensure that a program will execute consistently on all or any standard-conforming processors.

In some cases, this standard allows the provision of facilities that are not completely specified in the standard. These facilities are identified as **processor dependent**, and they shall be provided, with methods or semantics determined by the processor.

NOTE 1.1

> The processor should be accompanied by documentation that specifies the limits it imposes on the size and complexity of a program and the means of reporting when these limits are exceeded, that defines the additional forms and relationships it allows, and that defines the means of reporting the use of additional forms and relationships and the use of deleted or obsolescent forms. In this context, the use of a deleted form is the use of an additional form.
>
> The processor should be accompanied by documentation that specifies the methods or semantics of processor-dependent facilities.

1.5.1 Fortran 90 compatibility

Except as noted in this section, this standard is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539:1991, informally referred to as Fortran 90. Any standard-conforming Fortran 90 program that does not use one of the deleted features below remains standard-conforming under this standard. The following features present in Fortran 90 are not present in this standard (B.1):

1. Real and double precision DO variables,
2. Branching to an ENDIF statement from outside its IF construct,
3. PAUSE statement,
4. ASSIGN and assigned GOTO statements and assigned format specifiers, and
5. H edit descriptor.

NOTE 1.2

> Since a standard-conforming program is permitted to contain only forms and relationships described in this standard, any standard-conforming Fortran 90 program that contains any of these deleted features is not standard-conforming under this standard.

The following Fortran 90 features have different interpretations in this International Standard:

1. If the processor can distinguish between positive and negative real zero, the behavior of the SIGN intrinsic function when the second argument is negative real zero is changed by this standard.
2. This standard has more intrinsic procedures than did Fortran 90. Therefore, a standard-conforming Fortran 90 program may have a different interpretation under this standard if it invokes an external procedure having the same name as one of the new standard intrinsic procedures, unless that procedure is specified in an EXTERNAL statement or an interface body.

1.5.2 FORTRAN 77 compatibility

Except as noted in this section, the Fortran 95 Standard is an upward compatible extension to the earlier Fortran International Standard, ISO 1539:1980, informally referred to as FORTRAN 77. Any standard-conforming FORTRAN 77 program that does not use one of the deleted features listed in 1.5.1 remains standard conforming under the Fortran 95 Standard; however, see item (4) below regarding intrinsic procedures. The Fortran 95 Standard restricts the behavior for some features that were processor dependent in FORTRAN 77. Therefore, a standard-conforming FORTRAN 77 program that uses one of these processor-dependent features may have a different interpretation under the Fortran 95 Standard, yet remain a standard-conforming program. The following FORTRAN 77 features have different interpretations in the Fortran 95 Standard:

1. FORTRAN 77 permitted a processor to supply more precision derived from a real constant than can be represented in a real datum when the constant is used to initialize a data object of type double precision real in a DATA statement. The Fortran 95 Standard does not permit a processor this option.

(2) If a named variable that was not in a common block was initialized in a DATA statement and did not have the SAVE attribute specified, FORTRAN 77 left its SAVE attribute processor dependent. The Fortran 95 Standard specifies (5.2.10) that this named variable has the SAVE attribute.

(3) FORTRAN 77 required that the number of characters required by the input list was to be less than or equal to the number of characters in the record during formatted input. The Fortran 95 Standard specifies (9.4.4.4.2) that the input record is logically padded with blanks if there are not enough characters in the record, unless the PAD= specifier with the value 'NO' is specified in an appropriate OPEN statement.

(4) The Fortran 95 Standard has more intrinsic functions than did FORTRAN 77 and adds a few intrinsic subroutines. Therefore, a standard-conforming FORTRAN 77 program may have a different interpretation under the Fortran 95 Standard if it invokes an external procedure having the same name as one of the new standard intrinsic procedures, unless that procedure is specified in an EXTERNAL statement as recommended for nonintrinsic functions in appendix B of the FORTRAN 77 standard.

(5) A value of 0 for a list item in a formatted output statement will be formatted in a different form for some G edit descriptors. In addition, the Fortran 95 standard specifies how rounding of values will affect the output field form, but FORTRAN 77 did not address this issue: therefore, some FORTRAN 77 processors may produce a different output form than Fortran 95 processors for certain combinations of values and G edit descriptors.

(6) If the processor can distinguish between positive and negative real zero, the behavior of the SIGN intrinsic function when the second argument is negative real zero is changed by this standard.

1.6 Notation used in this standard

In this standard, "shall" is to be interpreted as a requirement; conversely, "shall not" is to be interpreted as a prohibition. Except where stated otherwise, such requirements and prohibitions apply to programs rather than processors.

1.6.1 Informative notes

Informative notes of explanation, rationale, examples, and other material are interspersed with the normative body of this standard. The informative material is identified by shading and is non-normative.

1.6.2 Syntax rules

Syntax rules are used to help describe the forms that Fortran lexical tokens, statements, and constructs may take. These syntax rules are expressed in a variation of Backus-Naur form (BNF) in which:

(1) Characters from the Fortran character set (3.1) are interpreted literally as shown, except where otherwise noted.

(2) Lower-case italicized letters and words (often hyphenated and abbreviated) represent general syntactic classes for which specific syntactic entities shall be substituted in actual statements.

Common abbreviations used in syntactic terms are:

stmt	for	statement	*attr*	for	attribute
expr	for	expression	*decl*	for	declaration
spec	for	specifier	*def*	for	definition
int	for	integer	*desc*	for	descriptor
arg	for	argument	*op*	for	operator

(3) The syntactic metasymbols used are:

is	introduces a syntactic class definition
or	introduces a syntactic class alternative
[ ]	encloses an optional item
[ ] ...	encloses an optionally repeated item which may occur zero or more times
■	continues a syntax rule

(4) Each syntax rule is given a unique identifying number of the form Rsnn, where s is a one- or two-digit section number and nn is a two-digit sequence number within that section. The syntax rules are distributed as appropriate throughout the text, and are referenced by number as needed. Some rules in Sections 2 and 3 are more fully described in later sections; in such cases, the section number s is the number of the later section where the rule is repeated.

(5) The syntax rules are not a complete and accurate syntax description of Fortran, and cannot be used to automatically generate a Fortran parser; where a syntax rule is incomplete, it is restricted by the corresponding constraints and text.

NOTE 1.3

An example of the use of the syntax rules is:

digit-string **is** *digit* [*digit*] ...

The following are examples of forms for a digit string allowed by the above rule:

digit
digit digit
digit digit digit digit
digit digit digit digit digit digit digit digit

When specific entities are substituted for digit, actual digit strings might be:

```
4
67
1999
10243852
```

1.6.3 Assumed syntax rules

In order to minimize the number of additional syntax rules and convey appropriate constraint information, the following rules are assumed. The letters "*xyz*" stand for any legal syntactic class phrase:

xyz-list **is** *xyz* [, *xyz*] ...

xyz-name **is** *name*

scalar-xyz **is** *xyz*

Constraint: *scalar-xyz* shall be scalar.

1.6.4 Syntax conventions and characteristics

(1) Any syntactic class name ending in "*-stmt*" follows the source form statement rules: it shall be delimited by end-of-line or semicolon, and may be labeled unless it forms part of another statement (such as an IF or WHERE statement). Conversely, everything considered to be a source form statement is given a "*-stmt*" ending in the syntax rules.

(2) The rules on statement ordering are described rigorously in the definition of *program-unit* (R202). Expression hierarchy is described rigorously in the definition of *expr* (R723).

(3) The suffix "*-spec*" is used consistently for specifiers, such as input/output statement specifiers. It also is used for type declaration attribute specifications (for example, "*array-spec*" in R513), and in a few other cases.

(4) When reference is made to a type parameter, including the surrounding parentheses, the suffix "*-selector*" is used. See, for example, "*kind-selector*" (R506) and "*length-selector*" (R508).

(5) The term "*subscript*" (for example, R617, R618, and R619) is used consistently in array definitions.

1.6.5 Text conventions

In the descriptive text, an English word equivalent of a BNF syntactic term is usually used. Specific statement keywords and attributes are identified in the text by the upper-case keyword, e.g., "END statement". Boldface words are used in the text where they are first defined with a specialized meaning. Obsolescent features (1.7) are shown in a distinguishing type size.

NOTE 1.4

This sentence is an example of the size used for obsolescent features.

1.7 Deleted and obsolescent features

This standard protects the users' investment in existing software by including all but five of the language elements of Fortran 90 that are not processor dependent. This standard identifies two categories of outmoded features. There are five in the first category, **deleted features**, which consists of features considered to have been redundant in FORTRAN 77 and largely unused in Fortran 90. Those in the second category, **obsolescent features**, are considered to have been redundant in Fortran 90, but are still frequently used.

1.7.1 Nature of deleted features

(1) Better methods existed in FORTRAN 77.

(2) These features are not included in this revision of Fortran.

1.7.2 Nature of obsolescent features

(1) Better methods existed in Fortran 90.

(2) It is recommended that programmers should use these better methods in new programs and convert existing code to these methods.

(3) These features are identified in the text of this document by a distinguishing type font (1.6.5).

(4) If the use of these features has become insignificant in Fortran programs, future Fortran standards committees should consider deleting them from the next revision.

(5) The next Fortran standards committee should consider for deletion only those language features that appear in the list of obsolescent features.

(6) Processors supporting the Fortran language should support these features as long as they continue to be used widely in Fortran programs.

1.8 Modules

This standard provides facilities that encourage the design and use of modular and reusable software. Data and procedure definitions may be organized into program units, called modules, and made available to any other program unit. In addition to global data and procedure library facilities, modules provide a mechanism for defining data abstractions and certain language extensions. Modules are described in 11.3.

1.9 Normative references

The following standards contain provisions which, through reference in this standard, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 8601:1988, *Data elements and interchange formats—Information interchange—Representation of dates and times*.

ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange*.