



BSI Standards Publication

Programming languages — Fortran

Part 1: Base language

National foreword

This British Standard is the UK implementation of ISO/IEC 1539-1:2023. It supersedes BS ISO/IEC 1539-1:2018, which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee IST/5/-/5, Fortran Panel.

A list of organizations represented on this committee can be obtained on request to its committee manager.

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2024
Published by BSI Standards Limited 2024

ISBN 978 0 539 16211 0

ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 April 2024.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

Programming languages — Fortran —
Part 1:
Base language

Langages de programmation — Fortran —
Partie 1: Langage de base



Reference number
ISO/IEC 1539-1:2023(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2023



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2023

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Foreword	xii
Introduction	xiii
1 Scope	1
2 Normative references	2
3 Terms and definitions	3
4 Notation, conformance, and compatibility	30
4.1 Notation, symbols and abbreviated terms	30
4.1.1 Syntax rules	30
4.1.2 Constraints	31
4.1.3 Assumed syntax rules	31
4.1.4 Syntax conventions and characteristics	31
4.1.5 Text conventions	32
4.2 Conformance	32
4.3 Compatibility	33
4.3.1 Previous Fortran standards	33
4.3.2 New intrinsic procedures	33
4.3.3 Fortran 2018 compatibility	33
4.3.4 Fortran 2008 compatibility	34
4.3.5 Fortran 2003 compatibility	35
4.3.6 Fortran 95 compatibility	36
4.3.7 Fortran 90 compatibility	36
4.3.8 FORTRAN 77 compatibility	37
4.4 Deleted and obsolescent features	37
4.4.1 General	37
4.4.2 Nature of deleted features	37
4.4.3 Nature of obsolescent features	38
5 Fortran concepts	39
5.1 High level syntax	39
5.2 Program unit concepts	42
5.2.1 Program units and scoping units	42
5.2.2 Program	42
5.2.3 Procedure	42
5.2.4 Module	43
5.2.5 Submodule	43
5.3 Execution concepts	43
5.3.1 Statement classification	43
5.3.2 Statement order	43
5.3.3 The END statement	44
5.3.4 Program execution	44
5.3.5 Execution sequence	45

5.3.6	Image execution states	45
5.3.7	Termination of execution	46
5.4	Data concepts	46
5.4.1	Type	46
5.4.2	Data value	47
5.4.3	Data entity	47
5.4.4	Definition of objects and pointers	48
5.4.5	Reference	49
5.4.6	Array	49
5.4.7	Coarray	49
5.4.8	Established coarrays	50
5.4.9	Pointer	50
5.4.10	Allocatable variables	50
5.4.11	Storage	51
5.5	Fundamental concepts	51
5.5.1	Names and designators	51
5.5.2	Statement keyword	51
5.5.3	Other keywords	51
5.5.4	Association	51
5.5.5	Intrinsic	51
5.5.6	Operator	52
5.5.7	Companion processors	52
6	Lexical tokens and source form	53
6.1	Processor character set	53
6.1.1	Characters	53
6.1.2	Letters	53
6.1.3	Digits	53
6.1.4	Underscore	53
6.1.5	Special characters	53
6.1.6	Other characters	54
6.2	Low-level syntax	54
6.2.1	Tokens	54
6.2.2	Names	54
6.2.3	Constants	55
6.2.4	Operators	55
6.2.5	Statement labels	56
6.2.6	Delimiters	56
6.3	Source form	57
6.3.1	Program units, statements, and lines	57
6.3.2	Free source form	57
6.3.3	Fixed source form	58
6.4	Including source text	59
7	Types	61
7.1	Characteristics of types	61
7.1.1	The concept of type	61
7.1.2	Type classification	61
7.1.3	Set of values	61
7.1.4	Constants	61
7.1.5	Operations	61
7.2	Type parameters	62
7.3	Types, type specifiers, and values	63
7.3.1	Relationship of types and values to objects	63
7.3.2	Type specifiers	63
7.3.3	Type compatibility	65

7.4	Intrinsic types	66
7.4.1	Classification and specification	66
7.4.2	Intrinsic operations on intrinsic types	66
7.4.3	Numeric intrinsic types	66
7.4.4	Character type	70
7.4.5	Logical type	73
7.5	Derived types	73
7.5.1	Derived type concepts	73
7.5.2	Derived-type definition	74
7.5.3	Derived-type parameters	77
7.5.4	Components	79
7.5.5	Type-bound procedures	85
7.5.6	Final subroutines	88
7.5.7	Type extension	90
7.5.8	Derived-type values	92
7.5.9	Derived-type specifier	92
7.5.10	Construction of derived-type values	92
7.5.11	Derived-type operations and assignment	95
7.6	Other nonintrinsic types	95
7.6.1	Interoperable enumerations and enum types	95
7.6.2	Enumeration types	97
7.7	Binary, octal, and hexadecimal literal constants	99
7.8	Construction of array values	100
8	Attribute declarations and specifications	103
8.1	Attributes of procedures and data objects	103
8.2	Type declaration statement	103
8.3	Automatic data objects	105
8.4	Initialization	105
8.5	Attributes	105
8.5.1	Attribute specification	105
8.5.2	Accessibility attribute	106
8.5.3	ALLOCATABLE attribute	106
8.5.4	ASYNCHRONOUS attribute	106
8.5.5	BIND attribute for data entities	107
8.5.6	CODIMENSION attribute	107
8.5.7	CONTIGUOUS attribute	109
8.5.8	DIMENSION attribute	110
8.5.9	EXTERNAL attribute	114
8.5.10	INTENT attribute	114
8.5.11	INTRINSIC attribute	116
8.5.12	OPTIONAL attribute	116
8.5.13	PARAMETER attribute	116
8.5.14	POINTER attribute	117
8.5.15	PROTECTED attribute	117
8.5.16	SAVE attribute	118
8.5.17	RANK clause	118
8.5.18	TARGET attribute	119
8.5.19	VALUE attribute	119
8.5.20	VOLATILE attribute	119
8.6	Attribute specification statements	120
8.6.1	Accessibility statement	120
8.6.2	ALLOCATABLE statement	121
8.6.3	ASYNCHRONOUS statement	121
8.6.4	BIND statement	121
8.6.5	CODIMENSION statement	122

8.6.6	CONTIGUOUS statement	122
8.6.7	DATA statement	122
8.6.8	DIMENSION statement	124
8.6.9	INTENT statement	125
8.6.10	OPTIONAL statement	125
8.6.11	PARAMETER statement	125
8.6.12	POINTER statement	125
8.6.13	PROTECTED statement	126
8.6.14	SAVE statement	126
8.6.15	TARGET statement	126
8.6.16	VALUE statement	126
8.6.17	VOLATILE statement	127
8.7	IMPLICIT statement	127
8.8	IMPORT statement	129
8.9	NAMelist statement	131
8.10	Storage association of data objects	132
8.10.1	EQUIVALENCE statement	132
8.10.2	COMMON statement	134
8.10.3	Restrictions on common and equivalence	135
9	Use of data objects	136
9.1	Designator	136
9.2	Variable	136
9.3	Constants	137
9.4	Scalars	137
9.4.1	Substrings	137
9.4.2	Structure components	137
9.4.3	Coindexed named objects	139
9.4.4	Complex parts	139
9.4.5	Type parameter inquiry	139
9.5	Arrays	140
9.5.1	Order of reference	140
9.5.2	Whole arrays	140
9.5.3	Array elements and array sections	140
9.5.4	Simply contiguous array designators	144
9.6	Image selectors	144
9.7	Dynamic association	146
9.7.1	ALLOCATE statement	146
9.7.2	NULLIFY statement	150
9.7.3	DEALLOCATE statement	150
9.7.4	STAT= specifier	152
9.7.5	ERRMSG= specifier	153
10	Expressions and assignment	154
10.1	Expressions	154
10.1.1	Expression semantics	154
10.1.2	Form of an expression	154
10.1.3	Precedence of operators	158
10.1.4	Evaluation of operations	160
10.1.5	Intrinsic operations	161
10.1.6	Defined operations	167
10.1.7	Evaluation of operands	168
10.1.8	Integrity of parentheses	169
10.1.9	Type, type parameters, and shape of an expression	169
10.1.10	Conformability rules for elemental operations	171
10.1.11	Specification expression	171

10.1.12	Constant expression	173
10.2	Assignment	174
10.2.1	Assignment statement	174
10.2.2	Pointer assignment	179
10.2.3	Masked array assignment – WHERE	183
10.2.4	FORALL	186
11	Execution control	189
11.1	Executable constructs containing blocks	189
11.1.1	Blocks	189
11.1.2	Rules governing blocks	189
11.1.3	ASSOCIATE construct	190
11.1.4	BLOCK construct	191
11.1.5	CHANGE TEAM construct	193
11.1.6	CRITICAL construct	195
11.1.7	DO construct	196
11.1.8	IF construct and statement	204
11.1.9	SELECT CASE construct	205
11.1.10	SELECT RANK construct	208
11.1.11	SELECT TYPE construct	210
11.1.12	EXIT statement	213
11.2	Branching	213
11.2.1	Branch concepts	213
11.2.2	GO TO statement	213
11.2.3	Computed GO TO statement	214
11.3	CONTINUE statement	214
11.4	STOP and ERROR STOP statements	214
11.5	FAIL IMAGE statement	215
11.6	NOTIFY WAIT statement	215
11.7	Image execution control	216
11.7.1	Image control statements	216
11.7.2	Segments	217
11.7.3	SYNC ALL statement	218
11.7.4	SYNC IMAGES statement	219
11.7.5	SYNC MEMORY statement	220
11.7.6	SYNC TEAM statement	221
11.7.7	EVENT POST statement	222
11.7.8	EVENT WAIT statement	222
11.7.9	FORM TEAM statement	222
11.7.10	LOCK and UNLOCK statements	223
11.7.11	STAT= and ERRMSG= specifiers in image control statements	225
12	Input/output statements	228
12.1	Input/output concepts	228
12.2	Records	228
12.2.1	Definition of a record	228
12.2.2	Formatted record	228
12.2.3	Unformatted record	228
12.2.4	Endfile record	229
12.3	External files	229
12.3.1	External file concepts	229
12.3.2	File existence	229
12.3.3	File access	230
12.3.4	File position	232
12.3.5	File storage units	233
12.4	Internal files	234

12.5	File connection	234
12.5.1	Referring to a file	234
12.5.2	Connection modes	235
12.5.3	Unit existence	236
12.5.4	Connection of a file to a unit	236
12.5.5	Preconnection	237
12.5.6	OPEN statement	237
12.5.7	CLOSE statement	241
12.6	Data transfer statements	243
12.6.1	Form of input and output statements	243
12.6.2	Control information list	243
12.6.3	Data transfer input/output list	248
12.6.4	Execution of a data transfer input/output statement	250
12.6.5	Termination of data transfer statements	261
12.7	Waiting on pending data transfer	261
12.7.1	Wait operation	261
12.7.2	WAIT statement	261
12.8	File positioning statements	262
12.8.1	Syntax	262
12.8.2	BACKSPACE statement	263
12.8.3	ENDFILE statement	263
12.8.4	REWIND statement	263
12.9	FLUSH statement	264
12.10	File inquiry statement	264
12.10.1	Forms of the INQUIRE statement	264
12.10.2	Inquiry specifiers	265
12.10.3	Inquire by output list	271
12.11	Error, end-of-record, and end-of-file conditions	271
12.11.1	Occurrence of input/output conditions	271
12.11.2	Error conditions and the ERR= specifier	272
12.11.3	End-of-file condition and the END= specifier	272
12.11.4	End-of-record condition and the EOR= specifier	273
12.11.5	IOSTAT= specifier	273
12.11.6	IOMSG= specifier	274
12.12	Restrictions on input/output statements	274
13	Input/output editing	275
13.1	Format specifications	275
13.2	Explicit format specification methods	275
13.2.1	FORMAT statement	275
13.2.2	Character format specification	275
13.3	Form of a format item list	276
13.3.1	Syntax	276
13.3.2	Edit descriptors	276
13.3.3	Fields	278
13.4	Interaction between input/output list and format	278
13.5	Positioning by format control	280
13.6	Decimal symbol	280
13.7	Data edit descriptors	280
13.7.1	Purpose of data edit descriptors	280
13.7.2	Numeric editing	281
13.7.3	Logical editing	288
13.7.4	Character editing	288
13.7.5	Generalized editing	289
13.7.6	User-defined derived-type editing	290
13.8	Control edit descriptors	290

13.8.1	Position edit descriptors	290
13.8.2	Slash editing	291
13.8.3	Colon editing	292
13.8.4	SS, SP, and S editing	292
13.8.5	LZS, LZP and LZ editing	292
13.8.6	P editing	292
13.8.7	BN and BZ editing	293
13.8.8	RU, RD, RZ, RN, RC, and RP editing	293
13.8.9	DC and DP editing	293
13.9	Character string edit descriptors	293
13.10	List-directed formatting	294
13.10.1	Purpose of list-directed formatting	294
13.10.2	Values and value separators	294
13.10.3	List-directed input	294
13.10.4	List-directed output	296
13.11	Namelist formatting	298
13.11.1	Purpose of namelist formatting	298
13.11.2	Name-value subsequences	298
13.11.3	Namelist input	298
13.11.4	Namelist output	301
14	Program units	303
14.1	Main program	303
14.2	Modules	303
14.2.1	Module syntax and semantics	303
14.2.2	The USE statement and use association	304
14.2.3	Submodules	307
14.3	Block data program units	307
15	Procedures	309
15.1	Concepts	309
15.2	Procedure classifications	309
15.2.1	Procedure classification by reference	309
15.2.2	Procedure classification by means of definition	309
15.3	Characteristics	310
15.3.1	Characteristics of procedures	310
15.3.2	Characteristics of dummy arguments	310
15.3.3	Characteristics of function results	310
15.4	Procedure interface	311
15.4.1	Interface and abstract interface	311
15.4.2	Implicit and explicit interfaces	311
15.4.3	Specification of the procedure interface	312
15.5	Procedure reference	321
15.5.1	Syntax of a procedure reference	321
15.5.2	Actual arguments, dummy arguments, and argument association	323
15.5.3	Function reference	335
15.5.4	Subroutine reference	335
15.5.5	Resolving named procedure references	336
15.5.6	Resolving type-bound procedure references	338
15.6	Procedure definition	338
15.6.1	Intrinsic procedure definition	338
15.6.2	Procedures defined by subprograms	338
15.6.3	Definition and invocation of procedures by means other than Fortran	344
15.6.4	Statement function	344
15.7	Pure procedures	345
15.8	Simple procedures	347

15.9	Elemental procedures	348
15.9.1	Elemental procedure declaration and interface	348
15.9.2	Elemental function actual arguments and results	348
15.9.3	Elemental subroutine actual arguments	349
16	Intrinsic procedures and modules	350
16.1	Classes of intrinsic procedures	350
16.2	Arguments to intrinsic procedures	350
16.2.1	General rules	350
16.2.2	The shape of array arguments	351
16.2.3	Mask arguments	351
16.2.4	DIM arguments and reduction functions	351
16.3	Bit model	351
16.3.1	General	351
16.3.2	Bit sequence comparisons	352
16.3.3	Bit sequences as arguments to INT and REAL	352
16.4	Numeric models	352
16.5	Atomic subroutines	353
16.6	Collective subroutines	354
16.7	Standard generic intrinsic procedures	355
16.8	Specific names for standard intrinsic functions	360
16.9	Specifications of the standard intrinsic procedures	362
16.9.1	General	362
16.10	Standard intrinsic modules	457
16.10.1	General	457
16.10.2	The ISO_FORTRAN_ENV intrinsic module	458
17	Exceptions and IEEE arithmetic	465
17.1	Overview of IEEE arithmetic support	465
17.2	Derived types, constants, and operators defined in the modules	466
17.3	The exceptions	466
17.4	The rounding modes	469
17.5	Underflow mode	469
17.6	Halting	470
17.7	The floating-point modes and status	470
17.8	Exceptional values	470
17.9	IEEE arithmetic	470
17.10	Summary of the procedures	471
17.11	Specifications of the procedures	473
17.11.1	General	473
17.12	Examples	499
18	Interoperability with C	502
18.1	General	502
18.2	The ISO_C_BINDING intrinsic module	502
18.2.1	Summary of contents	502
18.2.2	Named constants and derived types in the module	502
18.2.3	Procedures in the module	503
18.3	Interoperability between Fortran and C entities	511
18.3.1	Interoperability of intrinsic types	511
18.3.2	Interoperability with C pointer types	512
18.3.3	Interoperability of enum types	512
18.3.4	Interoperability of derived types and C structure types	512
18.3.5	Interoperability of scalar variables	513
18.3.6	Interoperability of array variables	514
18.3.7	Interoperability of procedures and procedure interfaces	514

18.4	C descriptors	517
18.5	The source file ISO_Fortran_binding.h	517
18.5.1	Summary of contents	517
18.5.2	The CFI_dim_t structure type	517
18.5.3	The CFI_cdesc_t structure type	518
18.5.4	Macros and typedefs in ISO_Fortran_binding.h	519
18.5.5	Functions declared in ISO_Fortran_binding.h	521
18.6	Restrictions on C descriptors	529
18.7	Restrictions on formal parameters	529
18.8	Restrictions on lifetimes	529
18.9	Interoperation with C global variables	530
18.9.1	General	530
18.9.2	Binding labels for common blocks and variables	531
18.10	Interoperation with C functions	531
18.10.1	Definition and reference of interoperable procedures	531
18.10.2	Binding labels for procedures	532
18.10.3	Exceptions and IEEE arithmetic procedures	532
18.10.4	Asynchronous communication	533
19	Scope, association, and definition	534
19.1	Scopes, identifiers, and entities	534
19.2	Global identifiers	534
19.3	Local identifiers	535
19.3.1	Classes of local identifiers	535
19.3.2	Local identifiers that are the same as common block names	536
19.3.3	Function results	536
19.3.4	Components, type parameters, and bindings	536
19.3.5	Argument keywords	536
19.4	Statement and construct entities	537
19.5	Association	538
19.5.1	Name association	538
19.5.2	Pointer association	542
19.5.3	Storage association	545
19.5.4	Inheritance association	547
19.5.5	Establishing associations	547
19.6	Definition and undefinition of variables	548
19.6.1	Definition of objects and subobjects	548
19.6.2	Variables that are always defined	548
19.6.3	Variables that are initially defined	548
19.6.4	Variables that are initially undefined	549
19.6.5	Events that cause variables to become defined	549
19.6.6	Events that cause variables to become undefined	551
19.6.7	Variable definition context	553
19.6.8	Pointer association context	554
Annex A	(informative) Processor dependencies	555
Annex B	(informative) Deleted and obsolescent features	562
Annex C	(informative) Extended notes	566
Index		643

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

This fifth edition cancels and replaces the fourth edition (ISO/IEC 1539-1:2018), which has been technically revised.

The main changes are as follows:

- an array can have a **coarray** component;
- additional forms of declaration;
- additional edit descriptors;
- additional intrinsic procedures;
- conformance with ISO/IEC 60559:2020;
- other changes listed in the Introduction.

A list of all parts in the ISO/IEC 1539 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document comprises the specification of the base Fortran language, informally known as Fortran 2023. With the limitations noted in 4.3.3, the syntax and semantics of Fortran 2018 are contained entirely within Fortran 2023. Therefore, any standard-conforming Fortran 2018 program not affected by such limitations is a standard-conforming Fortran 2023 program. New features of Fortran 2023 can be compatibly incorporated into such Fortran 2018 programs, with any exceptions indicated in the text of this document.

Fortran 2023 contains several extensions to Fortran 2018; these are listed below.

- Source form:
The maximum length of a line in free form source has been increased. The maximum length of a statement has been increased. The limit on the number of continuation lines has been removed.
- Data declaration:
A data object with a coarray component can be an array or allocatable. BIND(C) ENUM are now referred to as interoperable enumerations, and noninteroperable enumeration types are available. An interoperable enumeration can be given a type name. TYPEOF and CLASSOF type specifiers can be used to declare one or more entities to have the same type and type parameters as another entity. A PUBLIC namelist group can have a PRIVATE namelist group object. The DIMENSION attribute can be declared with a syntax that does not depend on the rank (8.5.8, 8.5.17).
- Data usage and computation:
Binary, octal, and hexadecimal literal constants can be used in additional contexts. A deferred-length allocatable errmsg-variable is allocated by the processor to the length of the explanatory message. An ALLOCATE statement can specify the bounds of an array allocation with array expressions. A pointer assignment statement can specify lower bounds or rank remapping with array expressions. Arrays can be used to specify multiple subscripts or subscript triplets (9.5.3.2). Conditional expressions provide selective evaluation of subexpressions.
- Input/output:
The AT edit descriptor provides output of character values with trailing blanks trimmed. The LEADING_ZERO= specifier in the OPEN and WRITE statements, and the LZF, LZS and LZ control edit descriptors, provide control of optional leading zeros during formatted output. A deferred-length allocatable iomsg-variable is allocated by the processor to the length of the explanatory message. A deferred-length allocatable scalar io-unit in a WRITE statement is allocated by the processor to the length of the record to be written.
- Execution control:
The REDUCE locality specifier for the DO CONCURRENT construct specifies reduction variables for the loop. The NOTIFY WAIT statement, NOTIFY= specifier on an image selector, and the NOTIFY_TYPE from the intrinsic module ISO_FORTRAN_ENV provide one-sided data-oriented synchronization between images.
- Intrinsic procedures:
The intrinsic functions ACOSD, ASIND, ATAND, ATAN2D, COSD, SIND, and TAND are trigonometric functions in which angles are specified in degrees. The intrinsic functions ACOSPI, ASINPI, ATANPI, ATAN2PI, COSPI, SINPI, and TANPI are trigonometric functions in which angles are specified in half-revolutions (that is, as multiples of π). The intrinsic function SELECTED_LOGICAL_KIND returns kind type parameter values for type logical. The intrinsic subroutine SPLIT parses a string into tokens, one at a time. The intrinsic subroutine SYSTEM_CLOCK supports more than one system clock for an image. The intrinsic subroutine TOKENIZE parses a string into tokens. When a deferred-length allocatable actual argument of an intrinsic procedure is assigned character data, it is allocated by the processor to the length of the data. Execution of a collective subroutine can be successful on an image even when an error condition occurs for the corresponding execution on another image.
- Intrinsic modules:
Additional named constants LOGICAL8, LOGICAL16, LOGICAL32, LOGICAL64, and REAL16 have been added to the intrinsic module ISO_FORTRAN_ENV. The subroutines IEEE_GET_ROUNDING_MODE, IEEE_GET_UNDERFLOW_MODE, IEEE_SET_ROUNDING_MODE, and IEEE_SET_UNDERFLOW_MODE, from the intrinsic module IEEE_ARITHMETIC, are now considered to be pure and simple. The subroutines IEEE_GET_MODES, IEEE_GET_STATUS, IEEE_SET_MODES, and

`IEEE_SET_STATUS`, from the intrinsic module `IEEE_EXCEPTIONS`, are now considered to be pure and simple. The procedures `C_F_STRPOINTER` and `F_C_STRING` have been added to the intrinsic module `ISO_C_BINDING` to assist in the use of null-terminated strings. The subroutine `C_F_POINTER` in the intrinsic module `ISO_C_BINDING` has an extra optional dummy argument, `LOWER`, that specifies the lower bounds for `FPTR`.

- Changes to the intrinsic module `IEEE_ARITHMETIC` for conformance with ISO/IEC 60559:2020:
The new functions `IEEE_MAX`, `IEEE_MAX_MAG`, `IEEE_MIN`, and `IEEE_MIN_MAG` perform the operations maximum, maximumMagnitude, minimum, and minimumMagnitude in ISO/IEC 60559:2020. The functions `IEEE_MAX_NUM`, `IEEE_MAX_NUM_MAG`, `IEEE_MIN_NUM`, and `IEEE_MIN_NUM_MAG` now conform to the operations maximumNumber, maximumMagnitudeNumber, minimumNumber and minimumMagnitudeNumber in ISO/IEC 60559:2020; the changes affect the treatment of zeros and NaNs.
- Program units and procedures:
A procedure can be specified to be a [simple procedure](#); a [simple procedure](#) references or defines nonlocal variables only via its [dummy arguments](#). [Conditional arguments](#) provide [actual argument](#) selection in a [procedure reference](#).

This document is organized in 19 clauses, dealing with 8 conceptual areas. These 8 areas, and the clauses in which they are treated, are:

High/low level concepts	Clauses 4 , 5 , 6
Data concepts	Clauses 7 , 8 , 9
Computations	Clauses 10 , 16 , 17
Execution control	Clause 11
Input/output	Clauses 12 , 13
Program units	Clauses 14 , 15
Interoperability with C	Clause 18
Scoping and association rules	Clause 19

It also contains the following nonnormative material:

Processor dependencies	Annex A
Deleted and obsolescent features	Annex B
Extended notes	Annex C
Index	Index

Programming languages —Fortran —

Part 1: Base language

1 Scope

This document specifies the form and establishes the interpretation of programs expressed in the base Fortran language. The purpose of this document is to promote portability, reliability, maintainability, and efficient execution of Fortran programs for use on a variety of computing systems.

This document specifies

- the forms that a program written in the Fortran language can take,
- the rules for interpreting the meaning of a program and its data,
- the form of the input data to be processed by such a program, and
- the form of the output data resulting from the use of such a program.

Except where stated otherwise, requirements and prohibitions specified by this document apply to programs rather than processors.

This document does not specify

- the mechanism by which programs are transformed for use on computing systems,
- the operations required for setup and control of the use of programs on computing systems,
- the method of transcription of programs or their input or output data to or from a storage medium,
- the program and processor behavior when this document fails to establish an interpretation except for the processor detection and reporting requirements in items (2) to (10) of 4.2,
- the maximum number of [images](#), or the size or complexity of a program and its data that will exceed the capacity of any particular computing system or the capability of a particular processor,
- the mechanism for determining the number of [images](#) of a program,
- the physical properties of an [image](#) or the relationship between [images](#) and the computational elements of a computing system,
- the physical properties of the representation of quantities and the method of rounding, approximating, or computing numeric values on a particular processor, except by reference to ISO/IEC 60559:2020 under conditions specified in Clause 17,
- the physical properties of input/output records, files, and units, or
- the physical properties and implementation of storage.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

ISO/IEC 9899:2018, *Programming languages—C*

ISO/IEC 10646, *Information technology—Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC/IEEE 60559:2020, *Information technology — Microprocessor Systems — Floating-Point arithmetic*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://www.electropedia.org/>

3.1

actual argument

entity that determines argument association

Note 1 to entry: See 15.5.2.3 and 15.5.2.4.

Note 2 to entry: An *actual-arg*, *consequent-arg*, or *variable* in a *defined assignment statement*, are all examples of actual arguments.

3.2

allocatable

having the *ALLOCATABLE* attribute

Note 1 to entry: See 8.5.3.

3.3

array

set of scalar data, all of the same type and type parameters, whose individual elements are arranged in a rectangular pattern

Note 1 to entry: See 8.5.8 and 9.5.

3.3.1

array element

scalar *subobject* (3.138) of an array that has the same type and type parameters as the array

Note 1 to entry: Array elements are described in 9.5.3.

3.3.2

array pointer

array with the *POINTER* attribute

Note 1 to entry: The *POINTER* attribute is described in 8.5.14.

3.3.3

array section

array *subobject* (3.138) designated by *array-section*, and which is itself an array

Note 1 to entry: Array sections are described in 9.5.3.4.

3.3.4

assumed-shape array

nonallocatable nonpointer *dummy argument* (3.59) array that takes its shape from its *effective argument* (3.60)

Note 1 to entry: Assumed-shape arrays are described in 8.5.8.3.

3.3.5

assumed-size array

dummy argument (3.59) array whose size is assumed from that of its *effective argument* (3.60)

Note 1 to entry: Assumed-size arrays are described in 8.5.8.5.

3.3.6

deferred-shape array

allocatable (3.2) array or *array pointer* (3.3.2)

Note 1 to entry: Deferred-shape arrays are described in 8.5.8.4.

3.3.7

explicit-shape array

array declared with an *explicit-shape-spec-list* or *explicit-shape-bounds-spec*, which specifies explicit values for the *bounds* (3.17) in each dimension of the array

Note 1 to entry: Explicit-shape arrays are described in 8.5.8.2.

3.4

ASCII character

character whose representation method corresponds to ISO/IEC 646:1991 (International Reference Version)

3.5

associate name

name of *construct entity* (3.35) associated with a selector of an ASSOCIATE, CHANGE TEAM, SELECT RANK, or SELECT TYPE construct

Note 1 to entry: See 11.1.3, 11.1.5, 11.1.10, and 11.1.11.

3.6

associating entity

entity that did not exist prior to a dynamically-established association

Note 1 to entry: Dynamically-established associations are described in 19.5.5.

3.7

association

inheritance association (3.7.4), *name association* (3.7.6), *pointer association* (3.7.7), or *storage association* (3.7.8)

3.7.1

argument association

association between an *effective argument* (3.60) and a *dummy argument* (3.59)

Note 1 to entry: Argument association is described in 15.5.2.

3.7.2

construct association

association between a selector and an *associate name* (3.5) in an ASSOCIATE, CHANGE TEAM, SELECT RANK, or SELECT TYPE construct

Note 1 to entry: See 11.1.3, 11.1.5, 11.1.10, 11.1.11, and 19.5.1.6.

3.7.3

host association

name association, other than argument association, between entities in a submodule or contained *scoping unit* (3.120) and entities in its host

Note 1 to entry: Host association is described in 19.5.1.4.

3.7.4**inheritance association**

association between the inherited components of an *extended type* (3.144.5) and the components of its *parent component* (3.30.2)

Note 1 to entry: Inheritance association is described in 19.5.4.

3.7.5**linkage association**

association between a variable or common block with the *BIND attribute* and a C global variable

Note 1 to entry: Linkage association is described in 18.9 and 19.5.1.5.

3.7.6**name association**

argument association (3.7.1), *construct association* (3.7.2), *host association* (3.7.3), *linkage association* (3.7.5), or *use association* (3.7.9)

Note 1 to entry: Name association is described in 19.5.1.

3.7.7**pointer association**

association between a *pointer* (3.104) and a procedure or a variable with the *TARGET attribute*

Note 1 to entry: Pointer association is described in 19.5.2.

3.7.8**storage association**

association between storage sequences

Note 1 to entry: Storage association is described in 19.5.3.

3.7.9**use association**

association between entities in a module and entities in a scoping unit or construct that references that module, as specified by a *USE statement*

Note 1 to entry: Use association is described in 14.2.2.

3.8**assumed-rank dummy data object**

dummy data object (3.59.1) that assumes the rank, shape, and size of its *effective argument* (3.60)

Note 1 to entry: Assumed-rank entities are described in 8.5.8.7.

3.9**assumed-type**

declared with a *TYPE(*)* type specifier

Note 1 to entry: See 7.3.2.

3.10**attribute**

property of an entity that determines its uses

Note 1 to entry: Attributes of procedures and data objects are described in 8.1.

3.11

automatic data object

nondummy *data object* (3.42) with a *type parameter* (3.144.12) or *array bound* (3.17) that depends on the value of a *specification expression* (3.128) that is not a *constant expression* (3.36)

Note 1 to entry: Automatic data objects are described in 8.3.

3.12

base object

object designated by the leftmost *part-name*

Note 1 to entry: Base objects are described in 9.4.2.

Note 2 to entry: This only applies to the *data-ref* syntax (R911).

3.13

binding

type-bound procedure (3.109.7) or *final subroutine* (3.69)

3.14

binding name

name given to a specific or generic *type-bound procedure* (3.109.7) in the type definition

Note 1 to entry: Type-bound procedures are described in 7.5.5.

3.15

binding label

default character value specifying the name by which a global entity with the **BIND** attribute is known to the *companion processor* (3.29)

Note 1 to entry: Binding labels are described in 18.10.2 and 18.9.2.

3.16

block

sequence of executable constructs formed by the syntactic class *block*

Note 1 to entry: A block is treated as a unit by the executable constructs described in 11.1.

3.17

bound

array bound

limit of a dimension of an *array* (3.3)

3.18

branch target statement

statement whose *statement label* (3.132) appears as a *label* in a **GO TO** statement, computed **GO TO** statement, *alt-return-spec*, **END=** specifier, **EOR=** specifier, or **ERR=** specifier

Note 1 to entry: A branch target statement shall be an *action-stmt*, *associate-stmt*, *end-associate-stmt*, *if-then-stmt*, *end-if-stmt*, *select-case-stmt*, *end-select-stmt*, *select-rank-stmt*, *end-select-rank-stmt*, *select-type-stmt*, *end-select-type-stmt*, *do-stmt*, *end-do-stmt*, *block-stmt*, *end-block-stmt*, *critical-stmt*, *end-critical-stmt*, *forall-construct-stmt*, *where-construct-stmt*, *end-function-stmt*, *end-mp-subprogram-stmt*, *end-program-stmt*, or *end-subroutine-stmt*. Branching is described in 11.2.1.

3.19**C address**

value of type `C_PTR` or `C_FUNPTR` from the intrinsic module `ISO_C_BINDING` identifying a location

Note 1 to entry: This is the concept that ISO/IEC 9899:2018 calls the address. This can apply to a variable or procedure.

3.20**C descriptor**

C structure of type `CFI_cdesc_t` defined in the source file `ISO_Fortran_binding.h`

Note 1 to entry: C descriptors and the source file `ISO_Fortran_binding.h` are described in 18.4 and 18.5.

3.21**character context**

within a character literal constant or within a character string edit descriptor

Note 1 to entry: Character literal constants are described in 7.4.4. Character string edit descriptors are described in 13.3.2.

3.22**characteristics**

properties used to determine compatibility or consistency

Note 1 to entry: A *dummy argument* (3.59) has the characteristic of being a *dummy data object* (3.59.1), *dummy procedure* (3.109.1), or an asterisk (alternate return indicator). A *dummy data object* (3.59.1) has the additional characteristics listed in 15.3.2.2. A *dummy procedure* (3.109.1) has the additional characteristics listed in 15.3.2.3.

A function result has the characteristics listed in 15.3.3. A procedure has the characteristics listed in 15.3.1. The characteristics of intrinsic procedures are listed in 16.9.

3.23**coarray**

component (3.30), or *variable* (3.151), that has nonzero *corank* (3.39)

3.23.1**established coarray**

coarray (3.23) that is accessible using an *image-selector*

Note 1 to entry: Established coarrays are described in 5.4.8.

3.24**cobound**

bound (limit) of a *codimension* (3.25)

3.25**codimension**

dimension of the pattern formed by a set of corresponding *coarrays* (3.23)

3.26**coindexed object**

data object (3.42) whose *designator* (3.56) includes an *image-selector*

Note 1 to entry: Image selectors are described in 9.6.

3.27

collating sequence

one-to-one mapping from a character set into the nonnegative integers

Note 1 to entry: Collating sequences are described in [7.4.4.4](#).

3.28

common block

block of physical storage specified by a [COMMON statement](#)

Note 1 to entry: COMMON blocks are described in [8.10.2](#).

3.28.1

blank common

unnamed common block

3.29

companion processor

processor-dependent mechanism by which global data and procedures can be referenced or defined

Note 1 to entry: Companion processors are described in [5.5.7](#).

3.30

component

part of a derived type, or of an object of derived type, defined by a *component-def-stmt*

Note 1 to entry: Components are described in [7.5.4](#).

3.30.1

direct component

one of the components, or one of the direct components of a nonpointer nonallocatable component

Note 1 to entry: See [7.5.1](#).

3.30.2

parent component

component of an *extended type* ([3.144.5](#)) whose type is that of the *parent type* ([3.144.10](#)) and whose components are *inheritance associated* ([3.7.4](#)) with the *inherited* ([3.84](#)) components of the parent type

Note 1 to entry: Inheritance and the parent component are described in [7.5.7.2](#).

3.30.3

potential subobject component

nonpointer component, or potential subobject component of a nonpointer component

Note 1 to entry: See [7.5.1](#).

3.30.4

subcomponent

<structure (3.136)> direct component (3.30.1) that is a *subobject* ([3.138](#)) of the structure

Note 1 to entry: See [9.4.2](#).

3.30.5

ultimate component

component that is of *intrinsic type* ([3.144.8](#)), a *pointer* ([3.104](#)), or *allocatable* ([3.2](#)); or an ultimate component of a nonpointer nonallocatable component of derived type

3.31**component order**

ordering of the nonparent components of a derived type that is used for *intrinsic* (3.90) formatted input/output and, where component keywords are not used, *structure constructors* (3.136.2)

Note 1 to entry: Component order is described in 7.5.4.7.

3.32**conformable**

having the same shape, or one being an array and the other being scalar

Note 1 to entry: This is a relationship between two data entities.

3.33**connected**

relationship between a *unit* (3.148) and a file: each is connected if and only if the unit refers to the file

Note 1 to entry: See 12.5.4.

3.34**constant**

data object (3.42) that has a value and which cannot be defined, redefined, or become undefined during execution of a program

Note 1 to entry: See 6.2.3 and 9.3.

3.34.1**literal constant**

constant that does not have a name

Note 1 to entry: A literal constant has the syntax *literal-constant* (R605), and is of intrinsic type (7.4).

3.34.2**named constant**

named *data object* (3.42) with the **PARAMETER** attribute

3.35**construct entity**

entity whose identifier has the scope of a construct

Note 1 to entry: The scoping of such entities is described in 19.1 and 19.4.

3.36**constant expression**

expression satisfying requirements that ensure its value is constant

Note 1 to entry: A constant expression shall satisfy the requirements in 10.1.12.

3.37**contiguous**

⟨array⟩ whose array elements, in order, are not separated by other data objects

Note 1 to entry: The requirements for contiguous are defined in 8.5.7.

3.38**contiguous**

⟨multi-part data object⟩ whose parts, in order, are not separated by other data objects

3.39

corank

number of *codimensions* (3.25) of a *coarray* (3.23), or zero for objects that are not coarrays

Note 1 to entry: See 5.4.7 and 8.5.6.

3.40

cosubscript

scalar integer expression in an image selector

Note 1 to entry: The syntax of an image selector is specified by the BNF rule *image-selector*(R926). The syntax of a cosubscript is specified by the BNF rule *cosubscript*(R927).

3.41

data entity

data object (3.42), result of the evaluation of an expression, or the result of the execution of a function reference

3.42

data object

object

constant, variable, or subobject of a constant

Note 1 to entry: See 7.1.4, 9.2, and 5.4.3.2.4.

3.43

decimal symbol

character that separates the whole and fractional parts in the decimal representation of a real number in a file

Note 1 to entry: See 13.6.

3.44

declaration

specification of attributes for various program entities Note 1 to entry: Often this involves specifying the type of a named data object or specifying the shape of a named array object.

3.45

default initialization

mechanism for automatically initializing pointer components to have a defined pointer association status, and nonpointer components to have a particular value

Note 1 to entry: Default initialization is described in 7.5.4.6.

3.46

default-initialized

⟨*subcomponent* (3.30.4)⟩ subject to a *default initialization* (3.45) specified in the type definition for that component

3.47

definable

capable of *definition* (3.53) and permitted to become *defined* (3.48)

3.48

defined

⟨*data object* (3.42)⟩ with a valid value

3.49

defined

⟨*pointer* (3.104)⟩ whose pointer association status is associated or disassociated

Note 1 to entry: Pointer association is described in 19.5.2.2.

3.50**defined assignment**

assignment defined by a procedure

Note 1 to entry: See [10.2.1.4](#) and [15.4.3.4.3](#).

3.51**defined input/output**

input/output defined by a procedure and accessed via a *defined-io-generic-spec*

Note 1 to entry: See syntax rule R1509, and [12.6.4.8](#).

3.52**defined operation**

operation defined by a procedure

Note 1 to entry: See [10.1.6.1](#) and [15.4.3.4.2](#).

3.53**definition**

⟨*data object* ([3.42](#))⟩ process by which the data object becomes defined

Note 1 to entry: Such events are listed in [19.6.5](#).

3.54**definition**

⟨derived type, interoperable enumeration, enumeration type, or procedure⟩ specification of the type, enumeration, or procedure

Note 1 to entry: See [7.5.2](#), [7.6.1](#), [7.6.2](#), and [15.6](#).

3.55**descendant**

submodule that extends a module or submodule, or that extends another descendant thereof

Note 1 to entry: This is a relationship between a module or submodule and a submodule. Submodules are described in [14.2.3](#).

3.56**designator**

name followed by zero or more component selectors, complex part selectors, array section selectors, array element selectors, image selectors, and substring selectors

Note 1 to entry: Designators are defined in [9.1](#).

3.56.1**complex part designator**

designator that designates the real or imaginary part of a complex *data object* ([3.42](#)), independently of the other part

Note 1 to entry: Complex parts are described in [9.4.4](#).

3.56.2**object designator****data object designator**

designator ([3.56](#)) for a *data object* ([3.42](#))

Note 1 to entry: An object name is a special case of an object designator.

3.56.3

procedure designator

designator (3.56) for a procedure

3.57

disassociated

⟨pointer association⟩ pointer association status of not being associated with any target and not being undefined

Note 1 to entry: Pointer association status is described in 19.5.2.2.

3.58

disassociated

⟨pointer⟩ whose pointer association status is disassociated

3.59

dummy argument

entity whose identifier appears in a *dummy-arg-list* or *dummy-arg-name-list* in a **FUNCTION**, **SUBROUTINE**, **ENTRY**, or *statement function* statement, or whose name can be used as an *argument keyword* (3.94.1) in a reference to an *intrinsic* (3.90) procedure or a procedure in an intrinsic module

3.59.1

dummy data object

dummy argument (3.59) that is a data object

3.59.2

dummy function

dummy procedure (3.109.1) that is a function

3.60

effective argument

entity that is argument-associated with a *dummy argument* (3.59)

Note 1 to entry: Argument association is described in 15.5.2.4.

3.61

effective item

scalar object treated as a single entity in input/output

Note 1 to entry: An effective item results from the application of the rules in 12.6.3 to an input/output list.

3.62

elemental

independent scalar application of an action or operation to elements of an array or corresponding elements of a set of conformable arrays and scalars, or possessing the capability of elemental operation

Note 1 to entry: Combination of scalar and array operands or arguments combine the scalar operand(s) with each element of the array operand(s).

3.62.1

elemental assignment

assignment that operates elementally

3.62.2

elemental operation

operation that operates elementally

3.62.3

elemental operator

operator in an elemental operation

3.62.4**elemental procedure**

procedure that can be used elementally

Note 1 to entry: User-defined elemental procedures are described in [15.9](#).

3.62.5**elemental reference**

reference to an elemental procedure with at least one array actual argument

3.62.6**elemental subprogram**

subprogram with the [ELEMENTAL](#) prefix

Note 1 to entry: See [15.9.1](#).

3.63**END statement**

end-block-data-stmt, *end-function-stmt*, *end-module-stmt*, *end-mp-subprogram-stmt*, *end-program-stmt*, *end-submodule-stmt*, or *end-subroutine-stmt*

3.64**explicit initialization**

initialization of a data object by a specification statement

Note 1 to entry: See [8.4](#) and [8.6.7](#).

3.65**extent**

number of elements in a single dimension of an *array* ([3.3](#))

3.66**external file**

file that exists in a medium external to the program

Note 1 to entry: See [12.3](#).

3.67**external unit****external input/output unit**

entity that can be *connected* ([3.33](#)) to an *external file* ([3.66](#))

Note 1 to entry: External units and their connection are described in [12.5.3](#) and [12.5.4](#).

3.68**file storage unit**

unit of storage in a *stream file* ([3.135](#)) or an unformatted *record file* ([3.116](#))

Note 1 to entry: File storage units are described in [12.3.5](#).

3.69**final subroutine**

subroutine whose name appears in a [FINAL statement](#) in a type definition, and which can be automatically invoked by the processor when an object of that type is finalized

Note 1 to entry: See [7.5.6](#).

3.70

finalizable

⟨type⟩ has a final subroutine or a nonpointer nonallocatable component of finalizable type

3.71

finalizable

⟨nonpointer data entity⟩ of finalizable type

3.72

finalization

process of calling final subroutines when certain events occur

Note 1 to entry: These events are listed in [7.5.6.3](#).

3.73

function

procedure that is invoked by an expression

3.74

function result

entity that returns the value of a function

Note 1 to entry: See [15.6.2.2](#).

3.75

generic identifier

sequence of tokens that identifies a generic set of procedures or operations

Note 1 to entry: See [15.4.3.4](#). In this context, an operation could be defined input/output or an assignment.

3.76

host instance

instance of the host procedure that supplies the host environment

Note 1 to entry: Instances are described in [15.6.2.4](#).

Note 2 to entry: This is only applicable to an *internal procedure* ([3.109.3](#)), or a *dummy procedure* ([3.109.1](#)) or *procedure pointer* ([3.104.2](#)) that is associated with an *internal procedure* ([3.109.3](#)).

3.77

host scoping unit

host

scoping unit ([3.120](#)) immediately surrounding another scoping unit, or the scoping unit extended by a submodule

3.78

IEEE infinity

ISO/IEC/IEEE 60559:2020 conformant infinite floating-point value

3.79

IEEE NaN

NaN

ISO/IEC/IEEE 60559:2020 conformant floating-point datum that does not represent a number

3.80

image

instance of a Fortran program

Note 1 to entry: See [5.3.4](#).

3.80.1**active image**

image (3.80) that has not failed or stopped

Note 1 to entry: Image execution states are described in 5.3.6.

3.80.2**failed image**

image (3.80) that has not initiated termination but which has ceased to participate in program execution

3.80.3**stopped image**

image (3.80) that has initiated *normal termination*

3.81**image index**

integer value identifying an *image* (3.80) within a *team* (3.142)

3.82**image control statement**

statement that affects the execution ordering between *images* (3.80)

Note 1 to entry: Image execution control is described in 11.7.

3.83**inclusive scope**

nonblock *scoping unit* (3.120) plus every *block scoping unit* (3.120.1) whose *host* (3.77) is that scoping unit or that is nested within such a block scoping unit

Note 1 to entry: That is, inclusive scope is the scope as if **BLOCK constructs** were not scoping units.

3.84**inherit**

acquire entities (*components* (3.30), *type-bound procedures* (3.109.7), and *type parameters* (3.144.12)) through type extension from the parent type

Note 1 to entry: Inheritance is described in 7.5.7.2.

3.85**inquiry function**

intrinsic (3.90) function, or function in an intrinsic module, whose result depends on the properties of one or more of its arguments instead of their values

3.86**interface**

(procedure) name, procedure characteristics, dummy argument names, binding label, and generic identifiers

Note 1 to entry: See 15.4.1.

3.86.1**abstract interface**

set of procedure characteristics with dummy argument names

3.86.2**explicit interface**

interface of a procedure that includes all the characteristics of the procedure and names for its dummy arguments except for asterisk dummy arguments

Note 1 to entry: See 15.4.2.

3.86.3

generic interface

set of procedure interfaces identified by a *generic identifier* (3.75)

3.86.4

implicit interface

interface of a procedure that is not an explicit interface

Note 1 to entry: See 15.4.2 and 15.4.3.8.

3.86.5

specific interface

interface (3.86) identified by a nongeneric name

3.87

interface block

abstract interface block (3.87.1), *generic interface block* (3.87.2), or *specific interface block* (3.87.3)

Note 1 to entry: Interface blocks are described in 15.4.3.2.

3.87.1

abstract interface block

interface block with the **ABSTRACT** keyword; collection of *interface bodies* that specify named *abstract interfaces* (3.86.1)

3.87.2

generic interface block

interface block with a *generic-spec*; collection of *interface bodies* and procedure statements that are being given that generic identifier

3.87.3

specific interface block

interface block with no *generic-spec* or **ABSTRACT** keyword; collection of *interface bodies* that specify the interfaces of procedures

3.88

interoperable

⟨Fortran entity⟩ equivalent to an entity defined by or definable by the *companion processor* (3.29)

Note 1 to entry: Interoperability between Fortran and C entities is described in 18.3.

3.89

interoperable

⟨C entity⟩ equivalent to an entity defined by or definable by the Fortran processor

3.90

intrinsic

type, procedure, module, assignment, operator, or input/output operation defined in this document and accessible without further definition or specification, or a procedure or module provided by a processor but not defined in this document

3.90.1

standard intrinsic

intrinsic, defined in this document

3.91**internal file**

character variable that is *connected* (3.33) to an *internal unit* (3.92)

Note 1 to entry: Internal files are described in 12.4. File connection is described in 12.5.4.

3.92**internal unit**

input/output unit (3.148) that is *connected* (3.33) to an *internal file* (3.91)

3.93**ISO 10646 character**

character whose representation method corresponds to UCS-4 in ISO/IEC 10646

3.94**keyword**

statement keyword, argument keyword, type parameter keyword, or component keyword

3.94.1**argument keyword**

word that identifies the corresponding *dummy argument* (3.59) in an actual argument list

Note 1 to entry: Argument correspondence is described in 15.5.2.1.

3.94.2**component keyword**

word that identifies a *component* (3.30) in a *structure constructor* (3.136.2)

3.94.3**statement keyword**

word that is part of the syntax of a statement

Note 1 to entry: Statement keywords are described in 5.5.2.

3.94.4**type parameter keyword**

word that identifies a *type parameter* (3.144.12) in a *type-param-spec*

3.95**lexical token**

keyword, name, literal constant other than a complex literal constant, operator, label, delimiter, comma, =, =>, :, ::, :, or %

Note 1 to entry: See 6.2.

3.96**line**

sequence of zero or more characters

3.97**main program**

program unit (3.113) that is not a *subprogram* (3.139), *module* (3.99), *submodule* (3.137), or block data program unit

Note 1 to entry: See 14.1.

3.98

masked array assignment

[assignment statement](#) in a [WHERE statement](#) or [WHERE construct](#)

Note 1 to entry: See [10.2.3](#).

3.99

module

[program unit](#) ([3.113](#)) that can contain, or access from another module, definitions that can be made accessible to other program units

Note 1 to entry: Modules are described in [14.2](#).

3.100

name

identifier of a program constituent, beginning with an alphabetic character and containing only alphanumeric characters and underscores

Note 1 to entry: The form of a name follows the rules given in [6.2.2](#).

3.101

operand

data value that is the subject of an operator

3.102

operator

intrinsic-operator, *defined-unary-op*, or *defined-binary-op*

Note 1 to entry: These are defined by the syntax rules R608, R1004, and R1024.

3.103

passed-object dummy argument

dummy argument of a *type-bound procedure* ([3.109.7](#)) or *procedure pointer* ([3.104.2](#)) component that becomes associated with the object through which the procedure is invoked

Note 1 to entry: This is described in [7.5.4.5](#).

3.104

pointer

data pointer ([3.104.1](#)) or *procedure pointer* ([3.104.2](#))

3.104.1

data pointer

data entity ([3.41](#)) with the **POINTER** attribute

Note 1 to entry: See [8.5.14](#).

3.104.2

procedure pointer

procedure with the **POINTER** attribute

3.104.3

local procedure pointer

procedure pointer ([3.104.2](#)) that is part of a *local variable* ([3.151.2](#)), or a named procedure pointer that is not a *dummy argument* ([3.59](#)) or accessed by use or host association

3.105**pointer assignment**

association of a pointer with a target, by execution of a [pointer assignment statement](#) or an intrinsic [assignment statement](#) for a derived-type object that has the pointer as a subobject

Note 1 to entry: The pointer assignment statement is described in [10.2.2](#). Derived-type intrinsic assignment is described in [10.2.1.2](#).

3.106**polymorphic**

⟨data entity⟩ able to be of differing *dynamic types* ([3.144.4](#)) during program execution

Note 1 to entry: Polymorphic data objects are declared with the CLASS type specifier ([7.3.2.3](#)).

3.107**polymorphic**

⟨function⟩ having a result that is a polymorphic data entity

3.108**preconnected**

connected ([3.33](#)) at the beginning of execution of the program

Note 1 to entry: Preconnection is described in [12.5.5](#).

3.109**procedure**

entity encapsulating an arbitrary sequence of actions that can be invoked directly during program execution

3.109.1**dummy procedure**

dummy argument ([3.59](#)) that is a procedure

Note 1 to entry: See [15.2.2.3](#).

3.109.2**external procedure**

procedure defined by an *external subprogram* ([3.139.1](#)) or by means other than Fortran

Note 1 to entry: The syntax of an external subprogram is defined by R503. See also [15.6.3](#).

3.109.3**internal procedure**

procedure defined by an *internal subprogram* ([3.139.2](#))

Note 1 to entry: The syntax of an internal subprogram is defined by R512.

3.109.4**module procedure**

procedure defined by a module subprogram, or a specific procedure provided by an intrinsic module

Note 1 to entry: The syntax of a module subprogram is defined by R1408.

3.109.5**pure procedure**

procedure declared or defined to be pure ([15.7](#))

3.109.6

simple procedure

procedure declared or defined to be simple

Note 1 to entry: Simple procedures are described in [15.8](#).

3.109.7

type-bound procedure

procedure that is bound to a derived type and referenced via an object of that type

Note 1 to entry: Type-bound procedures are described in [7.5.5](#).

3.110

processor

combination of a computing system and mechanism by which programs are transformed for use on that computing system

3.111

processor dependent

not completely specified in this document, having methods and semantics determined by the processor

Note 1 to entry: For example, the number of decimal digits displayed in list-directed output of a real value may vary across processors.

3.112

program

set of Fortran *program units* ([3.113](#)) and entities defined by means other than Fortran that includes exactly one *main program* ([3.97](#))

3.113

program unit

main program ([3.97](#)), *external subprogram* ([3.139.1](#)), *module* ([3.99](#)), *submodule* ([3.137](#)), or *block data program unit*

Note 1 to entry: See [5.2.1](#).

3.114

rank

number of array dimensions of a *data entity* ([3.41](#)) that is an array, or zero for a scalar entity

3.115

record

sequence of values or characters in a file

Note 1 to entry: See [12.2](#).

3.116

record file

file composed of a sequence of records

Note 1 to entry: See [12.1](#).

3.117

reference

data object reference ([3.117.1](#)), *procedure reference* ([3.117.4](#)), or *module reference* ([3.117.3](#))

3.117.1

data object reference

appearance of a *data object designator* ([3.56.2](#)) in a context requiring its value at that point during execution

3.117.2**function reference**

appearance of the *procedure designator* (3.56.3) for a function, or operator symbol for a *defined operation* (3.52), in a context requiring execution of the function during expression evaluation

Note 1 to entry: See 15.5.3.

3.117.3**module reference**

appearance of a module name in a *USE statement*

Note 1 to entry: See 14.2.2.

3.117.4**procedure reference**

appearance of a *procedure designator* (3.56.3), operator symbol, or assignment symbol in a context requiring execution of the procedure at that point during execution; or occurrence of defined input/output or derived-type *finalization* (3.72)

Note 1 to entry: Defined input/output is described in 12.6.4.8.

3.118**saved**

having the *SAVE attribute*

Note 1 to entry: The SAVE attribute is described in 8.5.16.

3.119**scalar**

data entity (3.41) that can be represented by a single value of the type and that is not an *array* (3.3)

3.120**scoping unit**

BLOCK construct, derived-type definition, *interface body*, *program unit* (3.113), or subprogram, excluding all nested scoping units in it

3.120.1**block scoping unit**

scoping unit of a *BLOCK construct*

3.121**segment**

maximal sequence of executions on an *image* (3.80) of statements other than *image control statements* (3.82)

Note 1 to entry: Segments are described in 11.7.2.

3.122**sequence**

set of elements ordered by a one-to-one correspondence with the numbers 1, 2, to n

3.123**sequence structure**

scalar *data object* (3.42) of a *sequence type* (3.124)

3.124**sequence type**

derived type with the *SEQUENCE attribute*

Note 1 to entry: Sequence types are described in 7.5.2.3.

3.124.1

character sequence type

sequence type with no *allocatable* (3.2) or *pointer* (3.104) *components* (3.30), and whose components are all default character or of another character sequence type

3.124.2

numeric sequence type

sequence type with no *allocatable* (3.2) or *pointer* (3.104) *components* (3.30), and whose components are all default complex, default integer, default logical, default real, double precision real, or of another numeric sequence type

3.125

shape

array dimensionality of a data entity, represented as a rank-one array whose size is the *rank* (3.114) of the data entity and whose elements are the extents of the data entity

Note 1 to entry: Thus the shape of a scalar data entity is an array with rank one and size zero.

3.126

simply contiguous

satisfying requirements that ensure it is contiguous

Note 1 to entry: An entity that is simply contiguous shall satisfy the requirements specified in 9.5.4.

Note 2 to entry: These requirements are simple ones which make it clear that a designator or variable designates a *contiguous* (3.37) array. Only an array designator or variable can be simply contiguous.

3.127

size

⟨array⟩ total number of elements in the *array* (3.3)

3.128

specification expression

expression satisfying requirements that make it suitable for use in specifications

Note 1 to entry: A specification expression shall satisfy the requirements specified in 10.1.11.

3.128.1

component specification expression

specification expression satisfying additional requirements that make it suitable for use in specifications in a *component definition statement*

Note 1 to entry: A component specification expression shall specify the additional requirements specified in 10.1.11.

3.129

specific name

name that is not a generic name

3.130

statement

sequence of one or more complete or partial lines satisfying a syntax rule that ends in *-stmt*

Note 1 to entry: See 6.3.

3.130.1

executable statement

end-function-stmt, *end-mp-subprogram-stmt*, *end-program-stmt*, *end-subroutine-stmt*, or statement that is a member of the syntactic class *executable-construct*, excluding those in the *block-specification-part* of a **BLOCK construct**

3.130.2**nonexecutable statement**

statement that is not an *executable statement* (3.130.1)

3.131**statement entity**

entity whose identifier has the scope of a statement or part of a statement

Note 1 to entry: See 19.1 and 19.4.

3.132**statement label****label**

unsigned positive number of up to five digits that refers to an individual statement

Note 1 to entry: Statement labels are described in 6.2.5.

3.133**storage sequence**

contiguous sequence of *storage units* (3.134)

3.134**storage unit**

character storage unit (3.134.1), *numeric storage unit* (3.134.2), *file storage unit* (3.68), or *unspecified storage unit* (3.134.3)

Note 1 to entry: Storage units are described in 19.5.3.2.

3.134.1**character storage unit**

unit of storage that holds a default character value

3.134.2**numeric storage unit**

unit of storage that holds a default real, default integer, or default logical value

3.134.3**unspecified storage unit**

unit of storage that holds a value that is not default character, default real, double precision real, default integer, default logical, or default complex

3.135**stream file**

file composed of a sequence of file storage units

Note 1 to entry: See 12.1.

3.136**structure**

scalar *data object* (3.42) of *derived type* (3.144.3)

3.136.1**structure component**

component (3.30) of a structure

3.136.2

structure constructor

syntax that specifies a structure value or creates such a value

Note 1 to entry: The syntax of a structure constructor is defined by *structure-constructor* (R756, 7.5.10).

3.137

submodule

program unit (3.113) that extends a *module* (3.99) or another submodule

Note 1 to entry: Submodules are described in 14.2.3.

3.138

subobject

portion of *data object* (3.42) that can be referenced and, if it is a *variable* (3.151), defined, independently of any other portion

Note 1 to entry: The conditions for a structure component being a subobject are specified in 9.4.2.

3.139

subprogram

function-subprogram (R1532) or *subroutine-subprogram* (R1537)

3.139.1

external subprogram

subprogram that is not contained in a *main program* (3.97), *module* (3.99), *submodule* (3.137), or another subprogram

3.139.2

internal subprogram

subprogram that is contained in a *main program* (3.97) or another subprogram

3.139.3

module subprogram

subprogram that is contained in a *module* (3.99) or *submodule* (3.137) but is not an internal subprogram

3.140

subroutine

procedure invoked by a *CALL statement*, by *defined assignment* (3.50), or by some operations on derived-type entities

3.140.1

atomic subroutine

intrinsic subroutine that performs an action on its ATOM argument atomically

Note 1 to entry: Atomic subroutines are described in 16.5.

3.140.2

collective subroutine

intrinsic subroutine that performs a calculation on a *team* (3.142) of images without requiring synchronization

Note 1 to entry: Collective subroutines are described in 16.6.

3.141**target**

entity that is *pointer associated* with a *pointer* (3.104), entity on the right-hand-side of a *pointer assignment statement*, or entity with the *TARGET attribute*

Note 1 to entry: Pointer association is described in 19.5.2.2. The pointer assignment statement is described in 10.2.2. The TARGET attribute is described in 8.5.18.

3.142**team**

ordered set of *images* (3.80) created by execution of a *FORM TEAM statement*, or the initial ordered set of all images

Note 1 to entry: The FORM TEAM statement is described in 11.7.9.

3.142.1**current team**

team specified by the most recently executed *CHANGE TEAM statement* of a *CHANGE TEAM construct* that has not completed execution, or initial team if no *CHANGE TEAM construct* is being executed

3.142.2**initial team**

team existing at the beginning of program execution, consisting of all images

3.142.3**parent team**

current team at time of execution of the *FORM TEAM statement* that created the team

Note 1 to entry: The initial team does not have a parent team.

3.142.4**sibling teams**

teams created by a single set of corresponding executions of the *FORM TEAM statement*

3.142.5**team number**

−1 which identifies the initial team, or positive integer that identifies a team among its *sibling teams* (3.142.4)

3.143**transformational function**

intrinsic function, or function in an intrinsic module, that is neither *elemental* (3.62) nor an *inquiry function* (3.85)

3.144**type****data type**

named category of data characterized by a set of values, a syntax for denoting these values, and a set of operations that interpret and manipulate the values

Note 1 to entry: See 7.1.

3.144.1**abstract type**

type with the *ABSTRACT attribute*

Note 1 to entry: The ABSTRACT attribute is described in 7.5.7.1.

3.144.2

declared type

type that a data entity is declared to have, either explicitly or implicitly

Note 1 to entry: See [7.3.2](#) and [10.1.9](#).

3.144.3

derived type

type defined by a derived-type definition or by an intrinsic module

Note 1 to entry: See [7.5](#).

3.144.4

dynamic type

type of a data entity at a particular point during execution of a program

Note 1 to entry: See [7.3.2.3](#) and [10.1.9](#).

3.144.5

extended type

type with the [EXTENDS](#) attribute

Note 1 to entry: The [EXTENDS](#) attribute is described in [7.5.7.1](#).

3.144.6

extensible type

type that can be extended using the [EXTENDS](#) clause

Note 1 to entry: See [7.5.7.1](#).

3.144.7

extension type

is the same type or is an extended type whose parent type is an extension type of the other type

Note 1 to entry: This is a relation of one type with respect to another.

3.144.8

intrinsic type

type defined by this document that is always accessible

Note 1 to entry: Intrinsic types are described in [7.4](#).

3.144.9

numeric type

one of the types integer, real, and complex

3.144.10

parent type

type named in the [EXTENDS](#) clause

Note 1 to entry: Only an extended type has a parent type.

3.144.11

type compatible

compatibility of the type of one entity with respect to another for purposes such as *argument association* ([3.7.1](#)), *pointer association* ([3.7.7](#)), and allocation

Note 1 to entry: Type compatibility is described in [7.3.3](#).

3.144.12**type parameter**

value used to parameterize a type

Note 1 to entry: Type parameters are described in 7.2.

3.144.12.1**assumed type parameter**

length type parameter (3.144.12.4) that assumes the type parameter value from another entity

Note 1 to entry: The other entity is

- the selector for an *associate name* (3.5),
- the *constant-expr* for a *named constant* (3.34.2) of type character, or
- the *effective argument* (3.60) for a *dummy argument* (3.59).

3.144.12.2**deferred type parameter**

length type parameter (3.144.12.4) whose value can change during execution of a program and whose *type-param-value* is a colon

3.144.12.3**kind type parameter**

type parameter whose value is required to be defaulted or given by a *constant expression*

3.144.12.4**length type parameter**

type parameter whose value is permitted to be *assumed*, *deferred*, or given by a *specification expression* (3.128)

3.144.12.5**type parameter inquiry**

syntax (*type-param-inquiry*) that is used to inquire the value of a type parameter of a data object

Note 1 to entry: Type parameter enquiries are described in 9.4.5.

3.144.12.6**type parameter order**

ordering of the type parameters of a type used for derived-type specifiers

Note 1 to entry: Type parameter order is defined in 7.5.3.2. The syntax of a derived-type specifier is *derived-type-spec*, defined in 7.5.9.

3.145**ultimate argument**

nondummy entity with which a *dummy argument* (3.59) is associated via a chain of argument associations

Note 1 to entry: Argument association is described in 15.5.2.4.

3.146**undefined**

⟨data object⟩ without a valid value

3.147**undefined**

⟨pointer⟩ does not have a pointer association status of associated or disassociated

Note 1 to entry: Pointer association status is described in 19.5.2.2.

3.148

unit

input/output unit

means, specified by an *io-unit*, for referring to a file

Note 1 to entry: See 12.5.1.

3.149

unlimited polymorphic

able to have any *dynamic type* (3.144.4) during program execution

Note 1 to entry: See 7.3.2.3.

3.150

unsaved

without the *SAVE* attribute

Note 1 to entry: The *SAVE* attribute is described in 8.5.16.

3.151

variable

data entity (3.41) that can be *defined* (3.48) and redefined during execution of a program

3.151.1

event variable

scalar variable of type *EVENT_TYPE* from the intrinsic module *ISO_FORTRAN_ENV*

Note 1 to entry: See 16.10.2.10.

3.151.2

local variable

variable in a *scoping unit* (3.120) that is not a *dummy argument* (3.59) or part thereof, is not a global entity or part thereof, and is not an entity or part of an entity that is accessible outside that *scoping unit* (3.120)

3.151.3

lock variable

scalar variable of type *LOCK_TYPE* from the intrinsic module *ISO_FORTRAN_ENV*

Note 1 to entry: See 16.10.2.19.

3.151.4

notify variable

scalar variable of type *NOTIFY_TYPE* from the intrinsic module *ISO_FORTRAN_ENV*

Note 1 to entry: See 16.10.2.22.

3.151.5

team variable

scalar variable of type *TEAM_TYPE* from the intrinsic module *ISO_FORTRAN_ENV*

Note 1 to entry: See 16.10.2.34.

3.152

vector subscript

section-subscript that is an array

Note 1 to entry: See 9.5.3.4.3.

3.153

whole array

array component or array name without further qualification

Note 1 to entry: See [9.5.2](#).

4 Notation, conformance, and compatibility

4.1 Notation, symbols and abbreviated terms

4.1.1 Syntax rules

Syntax rules 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) with the following conventions.

- Characters from the Fortran character set ([6.1](#)) are interpreted literally as shown, except where otherwise noted.
- Lower-case italicized letters and words (often hyphenated and abbreviated) represent general syntactic classes for which particular syntactic entities shall be substituted in actual statements.

Common abbreviations used in syntactic terms are:

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

- 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 that may occur zero or more times
■	continues a syntax rule

- Each syntax rule is given a unique identifying number of the form Rsnn, where s is a one- or two-digit clause number and nn is a two-digit sequence number within that clause. The syntax rules are distributed as appropriate throughout the text, and are referenced by number as needed. Some rules in [Clauses 5](#) and [6](#) are more fully described in later clauses; in such cases, the clause number s is the number of the later clause where the rule is repeated.
- The syntax rules are not a complete and accurate syntax description of Fortran, and cannot be used to generate a Fortran parser automatically; where a syntax rule is incomplete, it is restricted by corresponding constraints and text.

NOTE

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

Some examples of *digit-string* are:

NOTE (cont.)

4
67
1999
10243852

4.1.2 Constraints

Each constraint is given a unique identifying number of the form Csmn, where s is a one- or two-digit clause number and nn is a two- or three-digit sequence number within that clause.

Often a constraint is associated with a particular syntax rule. Where that is the case, the constraint is annotated with the syntax rule number in parentheses. A constraint that is associated with a syntax rule constitutes part of the definition of the syntax term defined by the rule. It thus applies in all places where the syntax term appears.

Some constraints are not associated with particular syntax rules. The effect of such a constraint is similar to that of a restriction stated in the text, except that a processor is required to have the capability to detect and report violations of constraints (4.2). In some cases, a broad requirement is stated in text and a subset of the same requirement is also stated as a constraint. This indicates that a standard-conforming program is required to adhere to the broad requirement, but that a standard-conforming processor is required only to have the capability of diagnosing violations of the constraint.

4.1.3 Assumed syntax rules

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

R401 *xyz-list* is *xyz* [, *xyz*] ...

R402 *xyz-name* is *name*

R403 *scalar-xyz* is *xyz*

C401 (R403) *scalar-xyz* shall be scalar.

An explicit syntax rule for a term overrides an assumed rule.

4.1.4 Syntax conventions and characteristics

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.

The rules on statement ordering are described rigorously in the definition of *program-unit* (R502). Expression hierarchy is described rigorously in the definition of *expr* (R1023).

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 R814), and in a few other cases.

Where reference is made to a type parameter, including the surrounding parentheses, the suffix “-*selector*” is used. See, for example, “*kind-selector*” (R706) and “*length-selector*” (R722).

4.1.5 Text conventions

In descriptive text, an equivalent English word is frequently used in place of a syntactic term. Particular statements and attributes are identified in the text by an upper-case keyword, e.g., “END statement”. The descriptions of obsolescent features appear in a smaller type size.

NOTE

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

4.2 Conformance

A **program** (5.2.2) 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 document. A **program unit** (5.2.1) conforms to this document if it can be included in a **program** in a manner that allows the **program** to be standard conforming.

A Fortran **processor** shall:

- (1) execute 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) contain 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 constraints;
- (3) contain the capability to detect and report the use within a submitted **program unit** of a form or relationship that is not permitted by the numbered syntax rules or constraints, including the deleted features described in Annex B;
- (4) contain the capability to detect and report the use within a submitted **program unit** of an intrinsic type with a **kind type parameter** value not supported by the processor (7.4);
- (5) contain the capability to detect and report the use within a submitted **program unit** of source form or characters not permitted by Clause 6;
- (6) contain 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 Clause 19;
- (7) contain the capability to detect and report the use within a submitted **program unit** of a nonstandard intrinsic procedure (including one with the same name as a **standard intrinsic** procedure but with different requirements);
- (8) contain the capability to detect and report the use within a submitted **program unit** of a nonstandard intrinsic module;
- (9) contain the capability to detect and report the use within a submitted **program unit** of a procedure from a **standard intrinsic** module, if the procedure is not defined by this document or the procedure has different requirements from those specified by this document; and
- (10) contain 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** (13.2.1), 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 procedure unless the name has the **EXTERNAL attribute** (8.5.9) where it is used. A standard-conforming program shall not use nonstandard intrinsic procedures or modules 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 document or may include standard items that are not portable, such as **external procedures** defined by means

other than Fortran, conformance to this document does not ensure that a program will execute consistently on all or any standard-conforming processors.

The semantics of facilities that are identified as [processor dependent](#) are not completely specified in this document. They shall be provided, with methods or semantics determined by the processor.

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.

4.3 Compatibility

4.3.1 Previous Fortran standards

Table 4.3 lists the previous editions of the Fortran International Standard, along with their informal names.

Table 4.3 — Previous editions of the Fortran International Standard

Official designation	Informal name
ISO R 1539-1972	FORTRAN 66
ISO 1539-1980	FORTRAN 77
ISO/IEC 1539:1991	Fortran 90
ISO/IEC 1539-1:1997	Fortran 95
ISO/IEC 1539-1:2004	Fortran 2003
ISO/IEC 1539-1:2010	Fortran 2008
ISO/IEC 1539-1:2018	Fortran 2018

4.3.2 New intrinsic procedures

Each Fortran International Standard since ISO 1539:1980 (FORTRAN 77), defines more intrinsic procedures than the previous one. Therefore, a Fortran program conforming to an older standard might have a different interpretation under a newer standard if it invokes an external procedure having the same name as one of the new standard intrinsic procedures, unless that procedure is specified to have the [EXTERNAL attribute](#).

4.3.3 Fortran 2018 compatibility

Except as identified in this subclause, this document is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539-1:2018 (Fortran 2018). A standard-conforming Fortran 2018 program that does not use any feature identified in this subclause as being no longer permitted remains standard-conforming under this document.

Fortran 2018 allowed integer arguments to the intrinsic subroutine [SYSTEM_CLOCK](#) to be of any kind. This document requires integer arguments to [SYSTEM_CLOCK](#) to have a decimal exponent range at least as large as a default integer, and requires that all integer arguments in a reference to [SYSTEM_CLOCK](#) have the same kind type parameter.

Fortran 2018 permitted a variable in a [BLOCK construct](#) that was declared only by a [DATA statement](#) to be used before the [DATA statement](#). This document does not permit such usage.

Fortran 2018 permitted the POINTER and TARGET arguments to the intrinsic function [ASSOCIATED](#) to have

different rank; this document does not permit such usage.

The following Fortran 2018 features might have a different interpretation under this document.

- After an allocatable deferred length character variable is assigned a value by an `IOMSG=` or `ERRMSG=` clause, is the unit in an internal `WRITE` statement, or is an `INTENT (OUT)` argument in a reference to an intrinsic subroutine, that variable might be of shorter or longer length under this document than under Fortran 2018, since this document specifies intrinsic assignment semantics for these assignments.
- This document permits the intrinsic subroutine `SYSTEM_CLOCK` to use two or more clocks, with different characteristics based on the type and kind type parameters of its arguments. A program that invokes `SYSTEM_CLOCK` with different argument types or kinds in different references, could have a different interpretation under this document.
- The result of a reference to `IEEE_MAX_NUM`, `IEEE_MAX_NUM_MAG`, `IEEE_MIN_NUM`, or `IEEE_MIN_NUM_MAG` where one argument is a number and the other is a signaling NaN is specified to be the number in this document. Fortran 2018 specified that the result is a NaN.

4.3.4 Fortran 2008 compatibility

Except as identified in this subclause, and except for the deleted features noted in Clause B.2, this document is an upward compatible extension to ISO/IEC 1539-1:2010 (Fortran 2008). Any standard-conforming Fortran 2008 program that does not use any deleted features, and does not use any feature identified in this subclause as being no longer permitted, remains standard-conforming under this document.

Fortran 2008 specifies that the `IOSTAT=` variable shall be set to a processor-dependent negative value if the flush operation is not supported for the unit specified. This document specifies that the processor-dependent negative integer value shall be different from the named constants `IOSTAT_EOR` or `IOSTAT_END` from the intrinsic module `ISO_FORTRAN_ENV`.

Fortran 2008 permitted a noncontiguous array that was supplied as an `actual argument` corresponding to a `contiguous INTENT (INOUT) dummy argument` in one iteration of a `DO CONCURRENT` construct, without being previously defined in that iteration, to be defined in another iteration; this document does not permit this.

Fortran 2008 permitted a `pure` statement function to reference a `volatile` variable, and permitted a `local variable` of a `pure` subprogram or of a `BLOCK construct` within a `pure` subprogram to be `volatile` (provided it was not used); this document does not permit that.

Fortran 2008 permitted a `pure` function to have a result that has a `polymorphic` allocatable `ultimate component`; this document does not permit that.

Fortran 2008 permitted a `PROTECTED TARGET` variable accessed by `use association` to be used as an *initial-data-target*; this document does not permit that.

Fortran 2008 permitted a `named constant` to have declared type `LOCK_TYPE`, or have a noncoarray `potential subobject component` with declared type `LOCK_TYPE`; this document does not permit that.

Fortran 2008 permitted a `polymorphic` object to be finalized within a `DO CONCURRENT` construct; this document does not permit that.

Fortran 2008 permitted an unallocated `allocatable coarray` or `coindexed object` to be allocated by an `assignment statement`, provided it was scalar, nonpolymorphic, and had no `deferred type parameters`; this document does not permit that.

Fortran 2008 permitted the processor to use a common pseudorandom number generator for all `images`. This document requires separate seeds on each `image` for the pseudorandom number generator.

Fortran 2008 required `ACOSH` of a complex value to have the imaginary part nonnegative and had no requirement on the real part. This document requires `ACOSH` of a complex value to have a nonnegative real part and has no such requirement on the imaginary part.