



**BSI Standards Publication**

# Information technology — Programming languages — Fortran

---

Part 1: Base language

# National foreword

This British Standard is the UK implementation of ISO/IEC 1539-1:2018, incorporating corrigenda June 2021 and March 2023. It supersedes BS ISO/IEC 1539-1:2010, which is withdrawn.

The start and finish of text introduced or altered by corrigendum is indicated in the text by tags. Text altered by ISO/IEC corrigendum June 2021 is indicated in the text by AC1 AC1. Text altered by ISO/IEC corrigendum March 2023 is indicated in the text by AC2 AC2.

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 28458 4

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 November 2018.

## Amendments/corrigenda issued since publication

| Date           | Text affected                                                                   |
|----------------|---------------------------------------------------------------------------------|
| 31 August 2021 | Implementation of ISO/IEC corrigendum June 2021                                 |
| 30 April 2023  | Implementation of ISO/IEC corrigendum March 2023                                |
| 31 March 2024  | Corrections to National foreword, document publication year and corrigenda tags |

---

---

**Information technology —  
Programming languages — Fortran —  
Part 1:  
Base language**

*Technologies de l'information — Langages de programmation —  
Fortran —*

*Partie 1: Langage de base*



Reference number  
ISO/IEC 1539-1:2018(E)

**AVA Documents**  
Document Sharing Hub  
© ISO/IEC 2018



## **COPYRIGHT PROTECTED DOCUMENT**

© ISO/IEC 2018

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  
Fax: +41 22 749 09 47  
Email: [copyright@iso.org](mailto:copyright@iso.org)  
Website: [www.iso.org](http://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 . . . . .   | 24   |
| 4.1 Notation, symbols and abbreviated terms . . . . .  | 24   |
| 4.1.1 Syntax rules . . . . .                           | 24   |
| 4.1.2 Constraints . . . . .                            | 25   |
| 4.1.3 Assumed syntax rules . . . . .                   | 25   |
| 4.1.4 Syntax conventions and characteristics . . . . . | 25   |
| 4.1.5 Text conventions . . . . .                       | 26   |
| 4.2 Conformance . . . . .                              | 26   |
| 4.3 Compatibility . . . . .                            | 27   |
| 4.3.1 Previous Fortran standards . . . . .             | 27   |
| 4.3.2 New intrinsic procedures . . . . .               | 27   |
| 4.3.3 Fortran 2008 compatibility . . . . .             | 27   |
| 4.3.4 Fortran 2003 compatibility . . . . .             | 28   |
| 4.3.5 Fortran 95 compatibility . . . . .               | 29   |
| 4.3.6 Fortran 90 compatibility . . . . .               | 29   |
| 4.3.7 FORTRAN 77 compatibility . . . . .               | 29   |
| 4.4 Deleted and obsolescent features . . . . .         | 30   |
| 4.4.1 General . . . . .                                | 30   |
| 4.4.2 Nature of deleted features . . . . .             | 30   |
| 4.4.3 Nature of obsolescent features . . . . .         | 30   |
| 5 Fortran concepts . . . . .                           | 31   |
| 5.1 High level syntax . . . . .                        | 31   |
| 5.2 Program unit concepts . . . . .                    | 34   |
| 5.2.1 Program units and scoping units . . . . .        | 34   |
| 5.2.2 Program . . . . .                                | 34   |
| 5.2.3 Procedure . . . . .                              | 34   |
| 5.2.4 Module . . . . .                                 | 35   |
| 5.2.5 Submodule . . . . .                              | 35   |
| 5.3 Execution concepts . . . . .                       | 35   |
| 5.3.1 Statement classification . . . . .               | 35   |
| 5.3.2 Statement order . . . . .                        | 35   |
| 5.3.3 The END statement . . . . .                      | 36   |
| 5.3.4 Program execution . . . . .                      | 36   |
| 5.3.5 Execution sequence . . . . .                     | 37   |
| 5.3.6 Image execution states . . . . .                 | 37   |

|        |                                                       |    |
|--------|-------------------------------------------------------|----|
| 5.3.7  | Termination of execution . . . . .                    | 38 |
| 5.4    | Data concepts . . . . .                               | 38 |
| 5.4.1  | Type . . . . .                                        | 38 |
| 5.4.2  | Data value . . . . .                                  | 39 |
| 5.4.3  | Data entity . . . . .                                 | 39 |
| 5.4.4  | Definition of objects and pointers . . . . .          | 40 |
| 5.4.5  | Reference . . . . .                                   | 41 |
| 5.4.6  | Array . . . . .                                       | 41 |
| 5.4.7  | Coarray . . . . .                                     | 41 |
| 5.4.8  | Established coarrays . . . . .                        | 42 |
| 5.4.9  | Pointer . . . . .                                     | 42 |
| 5.4.10 | Allocatable variables . . . . .                       | 42 |
| 5.4.11 | Storage . . . . .                                     | 42 |
| 5.5    | Fundamental concepts . . . . .                        | 43 |
| 5.5.1  | Names and designators . . . . .                       | 43 |
| 5.5.2  | Statement keyword . . . . .                           | 43 |
| 5.5.3  | Other keywords . . . . .                              | 43 |
| 5.5.4  | Association . . . . .                                 | 43 |
| 5.5.5  | Intrinsic . . . . .                                   | 43 |
| 5.5.6  | Operator . . . . .                                    | 43 |
| 5.5.7  | Companion processors . . . . .                        | 44 |
| 6      | Lexical tokens and source form . . . . .              | 45 |
| 6.1    | Processor character set . . . . .                     | 45 |
| 6.1.1  | Characters . . . . .                                  | 45 |
| 6.1.2  | Letters . . . . .                                     | 45 |
| 6.1.3  | Digits . . . . .                                      | 45 |
| 6.1.4  | Underscore . . . . .                                  | 45 |
| 6.1.5  | Special characters . . . . .                          | 45 |
| 6.1.6  | Other characters . . . . .                            | 46 |
| 6.2    | Low-level syntax . . . . .                            | 46 |
| 6.2.1  | Tokens . . . . .                                      | 46 |
| 6.2.2  | Names . . . . .                                       | 46 |
| 6.2.3  | Constants . . . . .                                   | 47 |
| 6.2.4  | Operators . . . . .                                   | 47 |
| 6.2.5  | Statement labels . . . . .                            | 48 |
| 6.2.6  | Delimiters . . . . .                                  | 48 |
| 6.3    | Source form . . . . .                                 | 49 |
| 6.3.1  | Program units, statements, and lines . . . . .        | 49 |
| 6.3.2  | Free source form . . . . .                            | 49 |
| 6.3.3  | Fixed source form . . . . .                           | 50 |
| 6.4    | Including source text . . . . .                       | 51 |
| 7      | Types . . . . .                                       | 53 |
| 7.1    | Characteristics of types . . . . .                    | 53 |
| 7.1.1  | The concept of type . . . . .                         | 53 |
| 7.1.2  | Type classification . . . . .                         | 53 |
| 7.1.3  | Set of values . . . . .                               | 53 |
| 7.1.4  | Constants . . . . .                                   | 53 |
| 7.1.5  | Operations . . . . .                                  | 53 |
| 7.2    | Type parameters . . . . .                             | 54 |
| 7.3    | Types, type specifiers, and values . . . . .          | 54 |
| 7.3.1  | Relationship of types and values to objects . . . . . | 54 |
| 7.3.2  | Type specifiers and type compatibility . . . . .      | 55 |
| 7.4    | Intrinsic types . . . . .                             | 57 |
| 7.4.1  | Classification and specification . . . . .            | 57 |

|        |                                                            |     |
|--------|------------------------------------------------------------|-----|
| 7.4.2  | Intrinsic operations on intrinsic types . . . . .          | 57  |
| 7.4.3  | Numeric intrinsic types . . . . .                          | 57  |
| 7.4.4  | Character type . . . . .                                   | 61  |
| 7.4.5  | Logical type . . . . .                                     | 64  |
| 7.5    | Derived types . . . . .                                    | 64  |
| 7.5.1  | Derived type concepts . . . . .                            | 64  |
| 7.5.2  | Derived-type definition . . . . .                          | 65  |
| 7.5.3  | Derived-type parameters . . . . .                          | 69  |
| 7.5.4  | Components . . . . .                                       | 70  |
| 7.5.5  | Type-bound procedures . . . . .                            | 77  |
| 7.5.6  | Final subroutines . . . . .                                | 79  |
| 7.5.7  | Type extension . . . . .                                   | 81  |
| 7.5.8  | Derived-type values . . . . .                              | 83  |
| 7.5.9  | Derived-type specifier . . . . .                           | 83  |
| 7.5.10 | Construction of derived-type values . . . . .              | 84  |
| 7.5.11 | Derived-type operations and assignment . . . . .           | 86  |
| 7.6    | Enumerations and enumerators . . . . .                     | 86  |
| 7.7    | Binary, octal, and hexadecimal literal constants . . . . . | 87  |
| 7.8    | Construction of array values . . . . .                     | 88  |
| 8      | Attribute declarations and specifications . . . . .        | 91  |
| 8.1    | Attributes of procedures and data objects . . . . .        | 91  |
| 8.2    | Type declaration statement . . . . .                       | 91  |
| 8.3    | Automatic data objects . . . . .                           | 93  |
| 8.4    | Initialization . . . . .                                   | 93  |
| 8.5    | Attributes . . . . .                                       | 93  |
| 8.5.1  | Attribute specification . . . . .                          | 93  |
| 8.5.2  | Accessibility attribute . . . . .                          | 94  |
| 8.5.3  | ALLOCATABLE attribute . . . . .                            | 94  |
| 8.5.4  | ASYNCHRONOUS attribute . . . . .                           | 94  |
| 8.5.5  | BIND attribute for data entities . . . . .                 | 95  |
| 8.5.6  | CODIMENSION attribute . . . . .                            | 95  |
| 8.5.7  | CONTIGUOUS attribute . . . . .                             | 97  |
| 8.5.8  | DIMENSION attribute . . . . .                              | 98  |
| 8.5.9  | EXTERNAL attribute . . . . .                               | 101 |
| 8.5.10 | INTENT attribute . . . . .                                 | 101 |
| 8.5.11 | INTRINSIC attribute . . . . .                              | 103 |
| 8.5.12 | OPTIONAL attribute . . . . .                               | 104 |
| 8.5.13 | PARAMETER attribute . . . . .                              | 104 |
| 8.5.14 | POINTER attribute . . . . .                                | 104 |
| 8.5.15 | PROTECTED attribute . . . . .                              | 105 |
| 8.5.16 | SAVE attribute . . . . .                                   | 105 |
| 8.5.17 | TARGET attribute . . . . .                                 | 106 |
| 8.5.18 | VALUE attribute . . . . .                                  | 106 |
| 8.5.19 | VOLATILE attribute . . . . .                               | 106 |
| 8.6    | Attribute specification statements . . . . .               | 107 |
| 8.6.1  | Accessibility statement . . . . .                          | 107 |
| 8.6.2  | ALLOCATABLE statement . . . . .                            | 108 |
| 8.6.3  | ASYNCHRONOUS statement . . . . .                           | 108 |
| 8.6.4  | BIND statement . . . . .                                   | 108 |
| 8.6.5  | CODIMENSION statement . . . . .                            | 108 |
| 8.6.6  | CONTIGUOUS statement . . . . .                             | 109 |
| 8.6.7  | DATA statement . . . . .                                   | 109 |
| 8.6.8  | DIMENSION statement . . . . .                              | 111 |
| 8.6.9  | INTENT statement . . . . .                                 | 111 |
| 8.6.10 | OPTIONAL statement . . . . .                               | 112 |

|         |                                                             |     |
|---------|-------------------------------------------------------------|-----|
| 8.6.11  | PARAMETER statement . . . . .                               | 112 |
| 8.6.12  | POINTER statement . . . . .                                 | 112 |
| 8.6.13  | PROTECTED statement . . . . .                               | 113 |
| 8.6.14  | SAVE statement . . . . .                                    | 113 |
| 8.6.15  | TARGET statement . . . . .                                  | 113 |
| 8.6.16  | VALUE statement . . . . .                                   | 113 |
| 8.6.17  | VOLATILE statement . . . . .                                | 114 |
| 8.7     | IMPLICIT statement . . . . .                                | 114 |
| 8.8     | IMPORT statement . . . . .                                  | 116 |
| 8.9     | NAMelist statement . . . . .                                | 118 |
| 8.10    | Storage association of data objects . . . . .               | 119 |
| 8.10.1  | EQUIVALENCE statement . . . . .                             | 119 |
| 8.10.2  | COMMON statement . . . . .                                  | 121 |
| 8.10.3  | Restrictions on common and equivalence . . . . .            | 122 |
| 9       | Use of data objects . . . . .                               | 123 |
| 9.1     | Designator . . . . .                                        | 123 |
| 9.2     | Variable . . . . .                                          | 123 |
| 9.3     | Constants . . . . .                                         | 124 |
| 9.4     | Scalars . . . . .                                           | 124 |
| 9.4.1   | Substrings . . . . .                                        | 124 |
| 9.4.2   | Structure components . . . . .                              | 124 |
| 9.4.3   | Coindexed named objects . . . . .                           | 126 |
| 9.4.4   | Complex parts . . . . .                                     | 126 |
| 9.4.5   | Type parameter inquiry . . . . .                            | 126 |
| 9.5     | Arrays . . . . .                                            | 127 |
| 9.5.1   | Order of reference . . . . .                                | 127 |
| 9.5.2   | Whole arrays . . . . .                                      | 127 |
| 9.5.3   | Array elements and array sections . . . . .                 | 127 |
| 9.5.4   | Simply contiguous array designators . . . . .               | 130 |
| 9.6     | Image selectors . . . . .                                   | 131 |
| 9.7     | Dynamic association . . . . .                               | 132 |
| 9.7.1   | ALLOCATE statement . . . . .                                | 132 |
| 9.7.2   | NULLIFY statement . . . . .                                 | 136 |
| 9.7.3   | DEALLOCATE statement . . . . .                              | 136 |
| 9.7.4   | STAT= specifier . . . . .                                   | 138 |
| 9.7.5   | ERRMSG= specifier . . . . .                                 | 139 |
| 10      | Expressions and assignment . . . . .                        | 140 |
| 10.1    | Expressions . . . . .                                       | 140 |
| 10.1.1  | Expression semantics . . . . .                              | 140 |
| 10.1.2  | Form of an expression . . . . .                             | 140 |
| 10.1.3  | Precedence of operators . . . . .                           | 144 |
| 10.1.4  | Evaluation of operations . . . . .                          | 145 |
| 10.1.5  | Intrinsic operations . . . . .                              | 146 |
| 10.1.6  | Defined operations . . . . .                                | 153 |
| 10.1.7  | Evaluation of operands . . . . .                            | 154 |
| 10.1.8  | Integrity of parentheses . . . . .                          | 154 |
| 10.1.9  | Type, type parameters, and shape of an expression . . . . . | 155 |
| 10.1.10 | Conformability rules for elemental operations . . . . .     | 156 |
| 10.1.11 | Specification expression . . . . .                          | 156 |
| 10.1.12 | Constant expression . . . . .                               | 158 |
| 10.2    | Assignment . . . . .                                        | 159 |
| 10.2.1  | Assignment statement . . . . .                              | 159 |
| 10.2.2  | Pointer assignment . . . . .                                | 164 |
| 10.2.3  | Masked array assignment – WHERE . . . . .                   | 168 |

|         |                                                                    |     |
|---------|--------------------------------------------------------------------|-----|
| 10.2.4  | FORALL . . . . .                                                   | 170 |
| 11      | Execution control . . . . .                                        | 173 |
| 11.1    | Executable constructs containing blocks . . . . .                  | 173 |
| 11.1.1  | Blocks . . . . .                                                   | 173 |
| 11.1.2  | Rules governing blocks . . . . .                                   | 173 |
| 11.1.3  | ASSOCIATE construct . . . . .                                      | 174 |
| 11.1.4  | BLOCK construct . . . . .                                          | 175 |
| 11.1.5  | CHANGE TEAM construct . . . . .                                    | 177 |
| 11.1.6  | CRITICAL construct . . . . .                                       | 179 |
| 11.1.7  | DO construct . . . . .                                             | 180 |
| 11.1.8  | IF construct and statement . . . . .                               | 187 |
| 11.1.9  | SELECT CASE construct . . . . .                                    | 188 |
| 11.1.10 | SELECT RANK construct . . . . .                                    | 191 |
| 11.1.11 | SELECT TYPE construct . . . . .                                    | 193 |
| 11.1.12 | EXIT statement . . . . .                                           | 196 |
| 11.2    | Branching . . . . .                                                | 196 |
| 11.2.1  | Branch concepts . . . . .                                          | 196 |
| 11.2.2  | GO TO statement . . . . .                                          | 196 |
| 11.2.3  | Computed GO TO statement . . . . .                                 | 197 |
| 11.3    | CONTINUE statement . . . . .                                       | 197 |
| 11.4    | STOP and ERROR STOP statements . . . . .                           | 197 |
| 11.5    | FAIL IMAGE statement . . . . .                                     | 198 |
| 11.6    | Image execution control . . . . .                                  | 198 |
| 11.6.1  | Image control statements . . . . .                                 | 198 |
| 11.6.2  | Segments . . . . .                                                 | 199 |
| 11.6.3  | SYNC ALL statement . . . . .                                       | 200 |
| 11.6.4  | SYNC IMAGES statement . . . . .                                    | 201 |
| 11.6.5  | SYNC MEMORY statement . . . . .                                    | 202 |
| 11.6.6  | SYNC TEAM statement . . . . .                                      | 203 |
| 11.6.7  | EVENT POST statement . . . . .                                     | 204 |
| 11.6.8  | EVENT WAIT statement . . . . .                                     | 204 |
| 11.6.9  | FORM TEAM statement . . . . .                                      | 204 |
| 11.6.10 | LOCK and UNLOCK statements . . . . .                               | 205 |
| 11.6.11 | STAT= and ERRMSG= specifiers in image control statements . . . . . | 207 |
| 12      | Input/output statements . . . . .                                  | 210 |
| 12.1    | Input/output concepts . . . . .                                    | 210 |
| 12.2    | Records . . . . .                                                  | 210 |
| 12.2.1  | Definition of a record . . . . .                                   | 210 |
| 12.2.2  | Formatted record . . . . .                                         | 210 |
| 12.2.3  | Unformatted record . . . . .                                       | 210 |
| 12.2.4  | Endfile record . . . . .                                           | 211 |
| 12.3    | External files . . . . .                                           | 211 |
| 12.3.1  | External file concepts . . . . .                                   | 211 |
| 12.3.2  | File existence . . . . .                                           | 211 |
| 12.3.3  | File access . . . . .                                              | 212 |
| 12.3.4  | File position . . . . .                                            | 214 |
| 12.3.5  | File storage units . . . . .                                       | 215 |
| 12.4    | Internal files . . . . .                                           | 216 |
| 12.5    | File connection . . . . .                                          | 216 |
| 12.5.1  | Referring to a file . . . . .                                      | 216 |
| 12.5.2  | Connection modes . . . . .                                         | 217 |
| 12.5.3  | Unit existence . . . . .                                           | 218 |
| 12.5.4  | Connection of a file to a unit . . . . .                           | 218 |
| 12.5.5  | Preconnection . . . . .                                            | 219 |

|         |                                                               |     |
|---------|---------------------------------------------------------------|-----|
| 12.5.6  | OPEN statement . . . . .                                      | 219 |
| 12.5.7  | CLOSE statement . . . . .                                     | 223 |
| 12.6    | Data transfer statements . . . . .                            | 224 |
| 12.6.1  | Form of input and output statements . . . . .                 | 224 |
| 12.6.2  | Control information list . . . . .                            | 225 |
| 12.6.3  | Data transfer input/output list . . . . .                     | 229 |
| 12.6.4  | Execution of a data transfer input/output statement . . . . . | 231 |
| 12.6.5  | Termination of data transfer statements . . . . .             | 242 |
| 12.7    | Waiting on pending data transfer . . . . .                    | 242 |
| 12.7.1  | Wait operation . . . . .                                      | 242 |
| 12.7.2  | WAIT statement . . . . .                                      | 243 |
| 12.8    | File positioning statements . . . . .                         | 243 |
| 12.8.1  | Syntax . . . . .                                              | 243 |
| 12.8.2  | BACKSPACE statement . . . . .                                 | 244 |
| 12.8.3  | ENDFILE statement . . . . .                                   | 244 |
| 12.8.4  | REWIND statement . . . . .                                    | 245 |
| 12.9    | FLUSH statement . . . . .                                     | 245 |
| 12.10   | File inquiry statement . . . . .                              | 246 |
| 12.10.1 | Forms of the INQUIRE statement . . . . .                      | 246 |
| 12.10.2 | Inquiry specifiers . . . . .                                  | 246 |
| 12.10.3 | Inquire by output list . . . . .                              | 252 |
| 12.11   | Error, end-of-record, and end-of-file conditions . . . . .    | 253 |
| 12.11.1 | Occurrence of input/output conditions . . . . .               | 253 |
| 12.11.2 | Error conditions and the ERR= specifier . . . . .             | 253 |
| 12.11.3 | End-of-file condition and the END= specifier . . . . .        | 253 |
| 12.11.4 | End-of-record condition and the EOR= specifier . . . . .      | 254 |
| 12.11.5 | IOSTAT= specifier . . . . .                                   | 254 |
| 12.11.6 | IOMSG= specifier . . . . .                                    | 255 |
| 12.12   | Restrictions on input/output statements . . . . .             | 255 |
| 13      | Input/output editing . . . . .                                | 256 |
| 13.1    | Format specifications . . . . .                               | 256 |
| 13.2    | Explicit format specification methods . . . . .               | 256 |
| 13.2.1  | FORMAT statement . . . . .                                    | 256 |
| 13.2.2  | Character format specification . . . . .                      | 256 |
| 13.3    | Form of a format item list . . . . .                          | 257 |
| 13.3.1  | Syntax . . . . .                                              | 257 |
| 13.3.2  | Edit descriptors . . . . .                                    | 257 |
| 13.3.3  | Fields . . . . .                                              | 259 |
| 13.4    | Interaction between input/output list and format . . . . .    | 259 |
| 13.5    | Positioning by format control . . . . .                       | 260 |
| 13.6    | Decimal symbol . . . . .                                      | 261 |
| 13.7    | Data edit descriptors . . . . .                               | 261 |
| 13.7.1  | Purpose of data edit descriptors . . . . .                    | 261 |
| 13.7.2  | Numeric editing . . . . .                                     | 261 |
| 13.7.3  | Logical editing . . . . .                                     | 269 |
| 13.7.4  | Character editing . . . . .                                   | 269 |
| 13.7.5  | Generalized editing . . . . .                                 | 270 |
| 13.7.6  | User-defined derived-type editing . . . . .                   | 271 |
| 13.8    | Control edit descriptors . . . . .                            | 271 |
| 13.8.1  | Position edit descriptors . . . . .                           | 271 |
| 13.8.2  | Slash editing . . . . .                                       | 272 |
| 13.8.3  | Colon editing . . . . .                                       | 272 |
| 13.8.4  | SS, SP, and S editing . . . . .                               | 273 |
| 13.8.5  | P editing . . . . .                                           | 273 |
| 13.8.6  | BN and BZ editing . . . . .                                   | 273 |

|         |                                                                               |     |
|---------|-------------------------------------------------------------------------------|-----|
| 13.8.7  | RU, RD, RZ, RN, RC, and RP editing . . . . .                                  | 273 |
| 13.8.8  | DC and DP editing . . . . .                                                   | 274 |
| 13.9    | Character string edit descriptors . . . . .                                   | 274 |
| 13.10   | List-directed formatting . . . . .                                            | 274 |
| 13.10.1 | Purpose of list-directed formatting . . . . .                                 | 274 |
| 13.10.2 | Values and value separators . . . . .                                         | 274 |
| 13.10.3 | List-directed input . . . . .                                                 | 275 |
| 13.10.4 | List-directed output . . . . .                                                | 277 |
| 13.11   | Namelist formatting . . . . .                                                 | 278 |
| 13.11.1 | Purpose of namelist formatting . . . . .                                      | 278 |
| 13.11.2 | Name-value subsequences . . . . .                                             | 278 |
| 13.11.3 | Namelist input . . . . .                                                      | 279 |
| 13.11.4 | Namelist output . . . . .                                                     | 282 |
| 14      | Program units . . . . .                                                       | 283 |
| 14.1    | Main program . . . . .                                                        | 283 |
| 14.2    | Modules . . . . .                                                             | 283 |
| 14.2.1  | Module syntax and semantics . . . . .                                         | 283 |
| 14.2.2  | The USE statement and use association . . . . .                               | 284 |
| 14.2.3  | Submodules . . . . .                                                          | 287 |
| 14.3    | Block data program units . . . . .                                            | 287 |
| 15      | Procedures . . . . .                                                          | 289 |
| 15.1    | Concepts . . . . .                                                            | 289 |
| 15.2    | Procedure classifications . . . . .                                           | 289 |
| 15.2.1  | Procedure classification by reference . . . . .                               | 289 |
| 15.2.2  | Procedure classification by means of definition . . . . .                     | 289 |
| 15.3    | Characteristics . . . . .                                                     | 290 |
| 15.3.1  | Characteristics of procedures . . . . .                                       | 290 |
| 15.3.2  | Characteristics of dummy arguments . . . . .                                  | 290 |
| 15.3.3  | Characteristics of function results . . . . .                                 | 290 |
| 15.4    | Procedure interface . . . . .                                                 | 291 |
| 15.4.1  | Interface and abstract interface . . . . .                                    | 291 |
| 15.4.2  | Implicit and explicit interfaces . . . . .                                    | 291 |
| 15.4.3  | Specification of the procedure interface . . . . .                            | 292 |
| 15.5    | Procedure reference . . . . .                                                 | 300 |
| 15.5.1  | Syntax of a procedure reference . . . . .                                     | 300 |
| 15.5.2  | Actual arguments, dummy arguments, and argument association . . . . .         | 303 |
| 15.5.3  | Function reference . . . . .                                                  | 314 |
| 15.5.4  | Subroutine reference . . . . .                                                | 314 |
| 15.5.5  | Resolving named procedure references . . . . .                                | 314 |
| 15.5.6  | Resolving type-bound procedure references . . . . .                           | 316 |
| 15.6    | Procedure definition . . . . .                                                | 317 |
| 15.6.1  | Intrinsic procedure definition . . . . .                                      | 317 |
| 15.6.2  | Procedures defined by subprograms . . . . .                                   | 317 |
| 15.6.3  | Definition and invocation of procedures by means other than Fortran . . . . . | 322 |
| 15.6.4  | Statement function . . . . .                                                  | 323 |
| 15.7    | Pure procedures . . . . .                                                     | 323 |
| 15.8    | Elemental procedures . . . . .                                                | 325 |
| 15.8.1  | Elemental procedure declaration and interface . . . . .                       | 325 |
| 15.8.2  | Elemental function actual arguments and results . . . . .                     | 326 |
| 15.8.3  | Elemental subroutine actual arguments . . . . .                               | 326 |
| 16      | Intrinsic procedures and modules . . . . .                                    | 327 |
| 16.1    | Classes of intrinsic procedures . . . . .                                     | 327 |
| 16.2    | Arguments to intrinsic procedures . . . . .                                   | 327 |

|         |                                                                          |     |
|---------|--------------------------------------------------------------------------|-----|
| 16.2.1  | General rules . . . . .                                                  | 327 |
| 16.2.2  | The shape of array arguments . . . . .                                   | 328 |
| 16.2.3  | Mask arguments . . . . .                                                 | 328 |
| 16.2.4  | DIM arguments and reduction functions . . . . .                          | 328 |
| 16.3    | Bit model . . . . .                                                      | 328 |
| 16.3.1  | General . . . . .                                                        | 328 |
| 16.3.2  | Bit sequence comparisons . . . . .                                       | 329 |
| 16.3.3  | Bit sequences as arguments to INT and REAL . . . . .                     | 329 |
| 16.4    | Numeric models . . . . .                                                 | 329 |
| 16.5    | Atomic subroutines . . . . .                                             | 330 |
| 16.6    | Collective subroutines . . . . .                                         | 331 |
| 16.7    | Standard generic intrinsic procedures . . . . .                          | 332 |
| 16.8    | Specific names for standard intrinsic functions . . . . .                | 338 |
| 16.9    | Specifications of the standard intrinsic procedures . . . . .            | 339 |
| 16.9.1  | General . . . . .                                                        | 339 |
| 16.10   | Standard intrinsic modules . . . . .                                     | 426 |
| 16.10.1 | General . . . . .                                                        | 426 |
| 16.10.2 | The ISO_FORTRAN_ENV intrinsic module . . . . .                           | 426 |
| 17      | Exceptions and IEEE arithmetic . . . . .                                 | 433 |
| 17.1    | Overview of IEEE arithmetic support . . . . .                            | 433 |
| 17.2    | Derived types, constants, and operators defined in the modules . . . . . | 434 |
| 17.3    | The exceptions . . . . .                                                 | 434 |
| 17.4    | The rounding modes . . . . .                                             | 436 |
| 17.5    | Underflow mode . . . . .                                                 | 437 |
| 17.6    | Halting . . . . .                                                        | 437 |
| 17.7    | The floating-point modes and status . . . . .                            | 438 |
| 17.8    | Exceptional values . . . . .                                             | 438 |
| 17.9    | IEEE arithmetic . . . . .                                                | 438 |
| 17.10   | Summary of the procedures . . . . .                                      | 439 |
| 17.11   | Specifications of the procedures . . . . .                               | 441 |
| 17.11.1 | General . . . . .                                                        | 441 |
| 17.12   | Examples . . . . .                                                       | 466 |
| 18      | Interoperability with C . . . . .                                        | 468 |
| 18.1    | General . . . . .                                                        | 468 |
| 18.2    | The ISO_C_BINDING intrinsic module . . . . .                             | 468 |
| 18.2.1  | Summary of contents . . . . .                                            | 468 |
| 18.2.2  | Named constants and derived types in the module . . . . .                | 468 |
| 18.2.3  | Procedures in the module . . . . .                                       | 469 |
| 18.3    | Interoperability between Fortran and C entities . . . . .                | 474 |
| 18.3.1  | Interoperability of intrinsic types . . . . .                            | 474 |
| 18.3.2  | Interoperability with C pointer types . . . . .                          | 475 |
| 18.3.3  | Interoperability of derived types and C structure types . . . . .        | 475 |
| 18.3.4  | Interoperability of scalar variables . . . . .                           | 476 |
| 18.3.5  | Interoperability of array variables . . . . .                            | 477 |
| 18.3.6  | Interoperability of procedures and procedure interfaces . . . . .        | 477 |
| 18.4    | C descriptors . . . . .                                                  | 480 |
| 18.5    | The source file ISO_Fortran_binding.h . . . . .                          | 480 |
| 18.5.1  | Summary of contents . . . . .                                            | 480 |
| 18.5.2  | The CFI_dim_t structure type . . . . .                                   | 480 |
| 18.5.3  | The CFI_cdesc_t structure type . . . . .                                 | 481 |
| 18.5.4  | Macros and typedefs in ISO_Fortran_binding.h . . . . .                   | 482 |
| 18.5.5  | Functions declared in ISO_Fortran_binding.h . . . . .                    | 484 |
| 18.6    | Restrictions on C descriptors . . . . .                                  | 492 |
| 18.7    | Restrictions on formal parameters . . . . .                              | 492 |

|         |                                                                     |     |
|---------|---------------------------------------------------------------------|-----|
| 18.8    | Restrictions on lifetimes . . . . .                                 | 492 |
| 18.9    | Interoperation with C global variables . . . . .                    | 493 |
| 18.9.1  | General . . . . .                                                   | 493 |
| 18.9.2  | Binding labels for common blocks and variables . . . . .            | 494 |
| 18.10   | Interoperation with C functions . . . . .                           | 494 |
| 18.10.1 | Definition and reference of interoperable procedures . . . . .      | 494 |
| 18.10.2 | Binding labels for procedures . . . . .                             | 495 |
| 18.10.3 | Exceptions and IEEE arithmetic procedures . . . . .                 | 496 |
| 18.10.4 | Asynchronous communication . . . . .                                | 496 |
| 19      | Scope, association, and definition . . . . .                        | 497 |
| 19.1    | Scopes, identifiers, and entities . . . . .                         | 497 |
| 19.2    | Global identifiers . . . . .                                        | 497 |
| 19.3    | Local identifiers . . . . .                                         | 498 |
| 19.3.1  | Classes of local identifiers . . . . .                              | 498 |
| 19.3.2  | Local identifiers that are the same as common block names . . . . . | 499 |
| 19.3.3  | Function results . . . . .                                          | 499 |
| 19.3.4  | Components, type parameters, and bindings . . . . .                 | 499 |
| 19.3.5  | Argument keywords . . . . .                                         | 499 |
| 19.4    | Statement and construct entities . . . . .                          | 500 |
| 19.5    | Association . . . . .                                               | 501 |
| 19.5.1  | Name association . . . . .                                          | 501 |
| 19.5.2  | Pointer association . . . . .                                       | 505 |
| 19.5.3  | Storage association . . . . .                                       | 508 |
| 19.5.4  | Inheritance association . . . . .                                   | 510 |
| 19.5.5  | Establishing associations . . . . .                                 | 510 |
| 19.6    | Definition and undefinition of variables . . . . .                  | 511 |
| 19.6.1  | Definition of objects and subobjects . . . . .                      | 511 |
| 19.6.2  | Variables that are always defined . . . . .                         | 511 |
| 19.6.3  | Variables that are initially defined . . . . .                      | 511 |
| 19.6.4  | Variables that are initially undefined . . . . .                    | 512 |
| 19.6.5  | Events that cause variables to become defined . . . . .             | 512 |
| 19.6.6  | Events that cause variables to become undefined . . . . .           | 514 |
| 19.6.7  | Variable definition context . . . . .                               | 516 |
| 19.6.8  | Pointer association context . . . . .                               | 516 |
| Annex A | (informative) Processor dependencies . . . . .                      | 518 |
| Annex B | (informative) Deleted and obsolescent features . . . . .            | 524 |
| Annex C | (informative) Extended notes . . . . .                              | 528 |
| Index   | . . . . .                                                           | 600 |

## Foreword

- 1 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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.
- 2 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](http://www.iso.org/directives)).
- 3 Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](http://www.iso.org/patents)).
- 4 Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.
- 5 For an explanation on 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 the following URL: [www.iso.org/iso/foreword.html](http://www.iso.org/iso/foreword.html).
- 6 This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.
- 7 This fourth edition cancels and replaces the third edition (ISO 1539-1:2010), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 1539-1:2010/Cor. 1:2012, ISO/IEC 1539-1:2010/Cor. 2:2013, ISO/IEC 1539-1:2010/Cor. 3:2014, and ISO/IEC 1539-1:2010/Cor. 4:2016. The main changes compared to the previous edition are as follows:
  - The Technical Specifications ISO/IEC TS 29113:2012 and ISO/IEC TS 18508:2015 have been incorporated.
  - Support for IEEE floating-point arithmetic has been updated to ISO/IEC/IEEE 60559:2011.
- 8 A list of all parts in the ISO 1539 series can be found on the ISO website.
- 9 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](http://www.iso.org/members.html).

# Introduction

- 1 This document comprises the specification of the base Fortran language, informally known as Fortran 2018. With the limitations noted in 4.3.3, the syntax and semantics of Fortran 2008 are contained entirely within Fortran 2018. Therefore, any standard-conforming Fortran 2008 program not affected by such limitations is a standard-conforming Fortran 2018 program. New features of Fortran 2018 can be compatibly incorporated into such Fortran 2008 programs, with any exceptions indicated in the text of this document.
- 2 Fortran 2018 contains several extensions to Fortran 2008; these are listed below.
  - Data declaration:  
Constant properties of an object declared in its *entity-decl* can be used in its *initialization*. The **EQUIVALENCE** and **COMMON** statements and the **block data program unit** have been redundant since Fortran 90 and are now specified to be *obsolescent*. Diagnosis of the appearance of a **PROTECTED TARGET** variable accessed by *use association* as a *data-target* in a structure constructor is required.
  - Data usage and computation:  
The *declared type* of the value supplied for a *polymorphic allocatable component* in a *structure constructor* is no longer required to be the same as the *declared type* of the *component*. **FORALL** is now specified to be *obsolescent*. The type and kind of an implied DO variable in an *array constructor* or **DATA statement** can be specified within the constructor or statement. The **SELECT RANK construct** provides structured access to the elements of an *assumed-rank array*. Completing execution of a **BLOCK construct** can cause the *association status of a pointer* with the **PROTECTED attribute** to become undefined. The standard intrinsic operations **<**, **<=**, **>**, and **>=** (also known as **.LT.**, **.LE.**, **.GT.**, and **.GE.**) on IEEE numbers provide *compareSignaling{relation}* operations; the **=** and **/=** operations (also known as **.EQ.** and **.NE.**) provide *compareQuiet{relation}* operations. Finalization of an allocatable subobject during intrinsic assignment has been clarified. The *char-length* in an executable statement is no longer required to be a specification expression.
  - Input/output:  
The **SIZE= specifier** can be used with advancing input. It is no longer prohibited to open a file on more than one unit. The value assigned by the **RECL= specifier** in an **INQUIRE statement** has been standardized. The values assigned by the **POS=** and **SIZE=** specifiers in an **INQUIRE statement** for a unit that has pending asynchronous operations have been standardized. The **G0.d edit descriptor** can be used for list items of type **Integer**, **Logical**, and **Character**. The **D**, **E**, **EN**, and **ES** edit descriptors can have a field width of zero, analogous to the **F** edit descriptor. The exponent width *e* in a *data edit descriptor* can be zero, analogous to a field width of zero. Floating-point formatted input accepts hexadecimal-significand numbers that conform to ISO/IEC/IEEE 60559:2011. The **EX edit descriptor** provides hexadecimal-significand formatted output conforming to ISO/IEC/IEEE 60559:2011. An error condition occurs if unacceptable characters are presented for logical or numeric editing during execution of a formatted input statement.
  - Execution control:  
The **arithmetic IF statement** has been deleted. Labeled **DO loops** have been redundant since Fortran 90 and are now specified to be *obsolescent*. The **nonblock DO construct** has been deleted. The locality of a variable used in a **DO CONCURRENT construct** can be explicitly specified. The stop code in a **STOP** or **ERROR STOP** statement can be nonconstant. Output of the stop code and exception summary from the **STOP** and **ERROR STOP** statements can be controlled.
  - Intrinsic procedures and modules:  
In a reference to the intrinsic function **CMPLX** with an actual argument of type complex, no keyword is needed for a **KIND** argument. In references to the intrinsic functions **ALL**, **ANY**, **FINDLOC**, **IALL**, **IANY**, **IPARITY**, **MAXLOC**, **MAXVAL**, **MINLOC**, **MINVAL**, **NORM2**, **PARITY**, **PRODUCT**, **SUM**, and **THIS\_IMAGE**, the *actual argument* for **DIM** can be a present optional *dummy argument*. The new intrinsic function **COSHAPE** returns the coshape of a *coarray*. The new intrinsic function **OUT\_OF\_RANGE** tests whether a numeric value can be safely converted to a different type or kind. The new intrinsic subroutine **RANDOM\_INIT** establishes the initial state of the pseudorandom number generator used by **RANDOM\_NUMBER**. The new intrinsic function **REDUCE** performs user-specified array reductions. A processor is required to report use of a *nonstandard intrinsic* procedure, use of a *nonstandard intrinsic* module, and use of a nonstandard procedure from a *standard intrinsic* module. Integer and logical arguments to intrinsic proced-

ures and intrinsic module procedures that were previously required to be of default kind no longer have that requirement, except for `RANDOM_SEED`. Specific names for intrinsic functions are now deemed *obsolescent*. All standard procedures in the intrinsic module `ISO_C_BINDING`, other than `C_F_POINTER` [AC2] and `C_F_PROCPONTER`, [AC2] are now *pure*. The arguments to the intrinsic function `SIGN` can be of different kind. Nonpolymorphic pointer arguments to the intrinsic functions `EXTENDS_TYPE_OF` and `SAME_TYPE_AS` need not have defined pointer association status. The effects of invoking the intrinsic procedures `COMMAND_ARGUMENT_COUNT`, `GET_COMMAND`, and `GET_COMMAND_ARGUMENT`, on *images* other than *image* one, are no longer processor dependent. Access to error messages from the intrinsic subroutines `GET_COMMAND`, `GET_COMMAND_ARGUMENT`, and `GET_ENVIRONMENT_VARIABLE` is provided by an optional `ERRMSG` argument. The result of `NORM2` for a zero-sized array argument has been clarified.

- Program units and procedures:

The `IMPORT` statement can appear in a contained subprogram or `BLOCK` construct, and can restrict access via host association; diagnosis of violation of the `IMPORT` restrictions is required. The `GENERIC` statement can be used to declare generic interfaces. The number of procedure arguments is used in *generic resolution*. In a module, the *default accessibility* of entities accessed from another module can be controlled separately from the default accessibility of entities declared in the using module. An `IMPLICIT NONE` statement can require explicit declaration of the `EXTERNAL` attribute throughout a *scoping unit* and its contained *scoping units*. A defined operation need not specify `INTENT (IN)` for a dummy argument with the `VALUE` attribute. A defined assignment need not specify `INTENT (IN)` for the second dummy argument if it has the `VALUE` attribute. Procedures that are not declared with an asterisk *type-param-value*, including `ELEMENTAL` procedures, can be invoked recursively by default; the `RECURSIVE` keyword is advisory only. The `NON_RECURSIVE` keyword specifies that a procedure is not recursive. The `ERROR STOP` statement can appear in a *pure subprogram*. A dummy argument of a *pure function* is permitted in a variable definition context, if it has the `VALUE` attribute. A *coarray dummy argument* [AC2], or a *coarray ultimate component* of a dummy argument, [AC2] can be referenced or defined by another *image*.

- Features previously described by ISO/IEC TS 29113:2012:

A dummy data object can assume its rank from its effective argument. A dummy data object can assume the type from its effective argument, without having the ability to perform type selection. An *interoperable* procedure can have dummy arguments that are *assumed-type* and/or *assumed-rank*. An *interoperable* procedure can have dummy data objects that are *allocatable*, *assumed-shape*, *optional*, or *pointers*. The *character length* of a dummy data object of an *interoperable* procedure can be assumed. The argument to `C_LOC` can be a noninteroperable array. The `FPTR` argument to `C_F_POINTER` can be a noninteroperable array pointer. The argument to `C_FUNLOC` can be a noninteroperable procedure. The `FPTR` argument to `C_F_PROCPONTER` can be a noninteroperable procedure pointer. There is a new named constant `C_PTRDIFF_T` to provide interoperability with the C type `ptrdiff_t`.

Additionally to ISO/IEC TS 29113:2012, a scalar *actual argument* can be associated with an *assumed-type assumed-size dummy argument*, an *assumed-rank dummy data object* that is not associated with an *assumed-size array* can be used as the argument to the function `C_SIZEOF` from the intrinsic module `ISO_C_BINDING`, and the `type` argument to `CFI_establish` can have a positive value corresponding to an *interoperable C type*.

- Changes to the intrinsic modules `IEEE_ARITHMETIC`, `IEEE_EXCEPTIONS`, and `IEEE_FEATURES` for conformance with ISO/IEC/IEEE 60559:2011:

There is a new, optional, rounding mode `IEEE_AWAY`. The new type `IEEE_MODES_TYPE` encapsulates all floating-point modes. Features associated with subnormal numbers can be accessed with functions and types named ...`SUBNORMAL`... (the old ...`DENORMAL`... names remain). The new function `IEEE_FMA` performs fused multiply-add operations. The function `IEEE_INT` performs rounded conversions to integer type. The new functions `IEEE_MAX_NUM`, `IEEE_MAX_NUM_MAG`, `IEEE_MIN_NUM`, and `IEEE_MIN_NUM_MAG` calculate maximum and minimum numeric values. The new functions `IEEE_NEXT_DOWN` and `IEEE_NEXT_UP` return the adjacent machine numbers. The new functions `IEEE_QUIET_EQ`, `IEEE_QUIET_GE`, `IEEE_QUIET_GT`, `IEEE_QUIET_LE`, `IEEE_QUIET_LT`, and `IEEE_QUIET_NE` perform quiet comparisons. The new functions `IEEE_SIGNALING_EQ`, `IEEE_SIGNALING_GE`, `IEEE_SIGNALING_GT`, `IEEE_SIGNALING_LE`, `IEEE_SIGNALING_LT`, and `IEEE_SIGNALING_NE` perform signaling comparisons. The decimal rounding mode can be inquired and set independently of the binary rounding mode, using the `RADIX` argument to `IEEE_GET_ROUNDING_MODE` and `IEEE_SET_ROUNDING_MODE`. The new function `IEEE_-`

**REAL** performs rounded conversions to real type. The function **IEEE\_REM** now requires its arguments to have the same radix. The function **IEEE\_RINT** now has a **ROUND** argument to perform specific rounding. The new function **IEEE\_SIGNBIT** tests the sign bit of an IEEE number.

- Features previously described by ISO/IEC TS 18508:2015:

The **CRITICAL** statement has optional **ERRMSG=** and **STAT=** specifiers. The intrinsic subroutines **ATOMIC\_DEFINE** and **ATOMIC\_REF** have an optional **STAT** argument. The new intrinsic subroutines **ATOMIC\_ADD**, **ATOMIC\_AND**, **ATOMIC\_CAS**, **ATOMIC\_FETCH\_ADD**, **ATOMIC\_FETCH\_AND**, **ATOMIC\_FETCH\_OR**, **ATOMIC\_FETCH\_XOR**, **ATOMIC\_OR**, and **ATOMIC\_XOR** perform atomic operations. The new intrinsic functions **FAILED\_IMAGES** and **STOPPED\_IMAGES** return indices of images known to have **failed** or **stopped** respectively. The new intrinsic function **IMAGE\_STATUS** returns the image execution status of an image. The intrinsic subroutine **MOVE\_ALLOC** has optional **ERRMSG** and **STAT** arguments. The intrinsic functions **IMAGE\_INDEX** and **NUM\_IMAGES** have additional forms with a **TEAM** or **TEAM\_NUMBER** argument. The intrinsic function **THIS\_IMAGE** has an optional **TEAM** argument. The **EVENT POST** and **EVENT WAIT** statements, the intrinsic subroutine **EVENT\_QUERY**, and the type **EVENT\_TYPE** provide an event facility for one-sided segment ordering. The **CHANGE TEAM** construct, derived type **TEAM\_TYPE**, **FORM TEAM** and **SYNC TEAM** statements, intrinsic functions **GET\_TEAM** and **TEAM\_NUMBER**, and the **TEAM=** and **TEAM\_NUMBER=** specifiers on image selectors, provide a team facility for a subset of the program's images to act in concert as if it were the set of all images. This team facility allows an allocatable **coarray** to be allocated or deallocated on a subset of images. The new intrinsic subroutines **CO\_BROADCAST**, **CO\_MAX**, **CO\_MIN**, **CO\_REDUCE**, and **CO\_SUM** perform collective reduction operations on the images of the **current team**. The concept of **failed images**, the **FAIL IMAGE** statement, the **STAT=** specifier on image selectors, and the named constant **STAT\_FAILED\_IMAGE** from the intrinsic module **ISO\_FORTRAN\_ENV** provide support for fault-tolerant parallel execution.

- Changes to features previously described by ISO/IEC TS 18508:2015:

The **CHANGE TEAM** and **SYNC TEAM** statements, and the **TEAM=** specifier on image selectors, permit the team to be specified by an expression. The intrinsic functions **FAILED\_IMAGES** and **STOPPED\_IMAGES** have no restriction on the kind of their result. The name of the function argument to the intrinsic function **CO\_REDUCE** is **OPERATION** instead of **OPERATOR**; this argument is not required to be commutative. The named constant **STAT\_UNLOCKED\_FAILED\_IMAGE** from the intrinsic module **ISO\_FORTRAN\_ENV** indicates that a **lock variable** was locked by an image that **failed**. The **team number** for the **initial team** can be used in image selectors, and in the intrinsic functions **NUM\_IMAGES** and **IMAGE\_INDEX**. A **team variable** that appears in a **CHANGE TEAM** statement can no longer be defined or become undefined during execution of the **CHANGE TEAM** construct. All images of the **current team** are no longer required to execute the same **CHANGE TEAM** statement. A variable of type **TEAM\_TYPE** from the intrinsic module **ISO\_FORTRAN\_ENV** is not permitted to be a **coarray**. A variable of type **TEAM\_TYPE** from the intrinsic module **ISO\_FORTRAN\_ENV** can have a pointer component, and a **team variable** becomes undefined if assigned a value from another **image**. The intrinsic function **UCOBOUND** produces a value for the final upper **cobound** that is always relative to the **current team**. An **EXIT** statement can be used to complete execution of a **CHANGE TEAM** or **CRITICAL** construct.

- 3 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          |

- 4 It also contains the following nonnormative material:

|                                  |                         |
|----------------------------------|-------------------------|
| Processor dependencies           | Annex <a href="#">A</a> |
| Deleted and obsolescent features | Annex <a href="#">B</a> |
| Extended notes                   | Annex <a href="#">C</a> |
| Index                            | Index                   |

# Information technology — Programming languages — Fortran —

## Part 1: Base language

### 1 Scope

- 1 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.
- 2 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.
- 3 Except where stated otherwise, requirements and prohibitions specified by this document apply to programs rather than processors.
- 4 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/IEEE 60559:2011 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 (International Reference Version), *Information technology—ISO 7-bit coded character set for information interchange*

ISO/IEC 9899:2011, *Programming languages—C*

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

ISO/IEC/IEEE 60559:2011, *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 terminological databases for use in standardization at the following addresses:

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

### 3.1

#### **actual argument**

entity (R1524) that appears in a [procedure reference](#)

### 3.2

#### **allocatable**

having the [ALLOCATABLE](#) attribute (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 (8.5.8, 9.5)

#### 3.3.1

##### **array element**

scalar individual element of an array

#### 3.3.2

##### **array pointer**

array with the [POINTER](#) attribute (8.5.14)

#### 3.3.3

##### **array section**

array [subobject](#) designated by [array-section](#), and which is itself an array (9.5.3.3)

#### 3.3.4

##### **assumed-shape array**

nonallocatable nonpointer [dummy argument](#) array that takes its shape from its [effective argument](#) (8.5.8.3)

#### 3.3.5

##### **assumed-size array**

[dummy argument](#) array whose size is assumed from that of its [effective argument](#) (8.5.8.5)

#### 3.3.6

##### **deferred-shape array**

[allocatable](#) array or array pointer, declared with a [deferred-shape-spec-list](#) (8.5.8.4)

#### 3.3.7

##### **explicit-shape array**

array declared with an [explicit-shape-spec-list](#), which specifies explicit values for the [bounds](#) in each dimension of the array (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](#) associated with a selector of an [ASSOCIATE](#), [CHANGE TEAM](#), [SELECT RANK](#), or [SELECT TYPE](#) construct ([11.1.3](#), [11.1.5](#), [11.1.10](#), [11.1.11](#))

## 3.6

### **associating entity**

(in a dynamically-established association) the entity that did not exist prior to the establishment of the association ([19.5.5](#))

## 3.7

### **association**

[inheritance association](#), [name association](#), [pointer association](#), or [storage association](#).

### 3.7.1

#### **argument association**

association between an [effective argument](#) and a [dummy argument](#) ([15.5.2](#))

### 3.7.2

#### **construct association**

association between a selector and an [associate name](#) in an [ASSOCIATE](#), [CHANGE TEAM](#), [SELECT RANK](#), or [SELECT TYPE](#) construct ([11.1.3](#), [11.1.5](#), [11.1.10](#), [11.1.11](#), [19.5.1.6](#))

### 3.7.3

#### **host association**

[name association](#), other than [argument association](#), between entities in a submodule or contained [scoping unit](#) and entities in its host ([19.5.1.4](#))

### 3.7.4

#### **inheritance association**

association between the [inherited](#) components of an [extended type](#) and the components of its [parent component](#) ([19.5.4](#))

### 3.7.5

#### **linkage association**

association between a variable or common block with the [BIND attribute](#) and a C global variable ([18.9](#), [19.5.1.5](#))

### 3.7.6

#### **name association**

[argument association](#), [construct association](#), [host association](#), [linkage association](#), or [use association](#) ([19.5.1](#))

### 3.7.7

#### **pointer association**

association between a [pointer](#) and an entity with the [TARGET attribute](#) ([19.5.2](#))

### 3.7.8

#### **storage association**

association between storage sequences ([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](#) ([14.2.2](#))

## 3.8

### **assumed-rank dummy data object**

[dummy data object](#) that assumes the rank, shape, and size of its [effective argument](#) ([8.5.8.7](#))

**3.9****assumed-type**

declared with a `TYPE(*)` type specifier (7.3.2)

**3.10****attribute**

property of an entity that determines its uses (8.1)

**3.11****automatic data object**

nondummy [data object](#) with a [type parameter](#) or array [bound](#) that depends on the value of a *specification-expr* that is not a [constant expression](#) (8.3)

**3.12****base object**

*<data-ref>* object designated by the leftmost *part-name* (9.4.2)

**3.13****binding**

[type-bound procedure](#) or [final subroutine](#) (7.5.5)

**3.14****binding name**

name given to a specific or generic [type-bound procedure](#) in the type definition (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](#) (18.10.2, 18.9.2)

**3.16****block**

sequence of executable constructs formed by the syntactic class *block* and which 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](#) (8.5.8)

**3.18****branch target statement**

*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*, whose statement label appears as a *label* in a `GO TO` statement, computed `GO TO` statement, *alt-return-spec*, `END=` specifier, `EOR=` specifier, or `ERR=` specifier (11.2.1)

**3.19****C address**

*<variable or procedure>* value of type `C_PTR` or `C_FUNPTR` from the intrinsic module `ISO_C_BINDING` identifying the location

Note 1 to entry: This is the concept that ISO/IEC 9899:2011 calls the address.

## 3.20

### **C descriptor**

C structure of type [CFI\\_cdesc\\_t](#) defined in the source file `ISO_Fortran_binding.h` ([18.4](#), [18.5](#))

## 3.21

### **character context**

within a character literal constant ([7.4.4](#)) or within a character string edit descriptor ([13.3.2](#))

## 3.22

### **characteristics**

⟨[dummy argument](#)⟩ being a [dummy data object](#), [dummy procedure](#), or an asterisk (alternate return indicator)

## 3.23

### **characteristics**

⟨[dummy data object](#)⟩ properties listed in [15.3.2.2](#)

## 3.24

### **characteristics**

⟨[dummy procedure](#) or [dummy procedure pointer](#)⟩ properties listed in [15.3.2.3](#)

## 3.25

### **characteristics**

⟨function result⟩ properties listed in [15.3.3](#)

## 3.26

### **characteristics**

⟨procedure⟩ properties listed in [15.3.1](#)

## 3.27

### **coarray**

[data entity](#) that has nonzero corank ([5.4.7](#))

### 3.27.1

#### **established coarray**

[coarray](#) that is accessible using an *[image-selector](#)* ([5.4.8](#))

## 3.28

### **cobound**

bound (limit) of a [codimension](#) ([8.5.6](#))

## 3.29

### **codimension**

dimension of the pattern formed by a set of corresponding [coarrays](#) ([8.5.6](#))

## 3.30

### **coindexed object**

[data object](#) whose [designator](#) includes an *[image-selector](#)* (R924, [9.6](#))

## 3.31

### **collating sequence**

one-to-one mapping from a character set into the nonnegative integers ([7.4.4.4](#))

## 3.32

### **common block**

block of physical storage specified by a [COMMON statement](#) ([8.10.2](#))

### 3.32.1

#### **blank common**

unnamed common block

**3.33****companion processor**

processor-dependent mechanism by which global data and procedures may be referenced or defined (5.5.7)

**3.34****component**

part of a derived type, or of an object of derived type, defined by a *component-def-stmt* (7.5.4)

**3.34.1****direct component**

one of the components, or one of the direct components of a nonpointer nonallocatable component (7.5.1)

**3.34.2****parent component**

component of an *extended type* whose type is that of the *parent type* and whose components are *inheritance associated* with the *inherited* components of the *parent type* (7.5.7.2)

**3.34.3****potential subobject component**

nonpointer component, or potential subobject component of a nonpointer component (7.5.1)

**3.34.4****subcomponent**

⟨*structure*⟩ *direct component* that is a *subobject* of the *structure* (9.4.2)

**3.34.5****ultimate component**

component that is of *intrinsic type*, a pointer, or allocatable; or an ultimate component of a nonpointer nonallocatable component of derived type

**3.35****component order**

ordering of the nonparent components of a derived type that is used for *intrinsic* formatted input/output and *structure constructors* (where component keywords are not used) (7.5.4.7)

**3.36****conformable**

⟨of two *data entities*⟩ having the same shape, or one being an array and the other being scalar

**3.37****connected**

relationship between a *unit* and a file: each is connected if and only if the *unit* refers to the file (12.5.4)

**3.38****constant**

*data object* that has a value and which cannot be defined, redefined, or become undefined during execution of a program (6.2.3, 9.3)

**3.38.1****literal constant**

constant that does not have a name (R605, 7.4)

**3.38.2****named constant**

named *data object* with the *PARAMETER attribute* (8.5.13)

## 3.39

### **construct entity**

entity whose identifier has the scope of a construct (19.1, 19.4)

## 3.40

### **constant expression**

expression satisfying the requirements specified in 10.1.12, thus ensuring that its value is constant

## 3.41

### **contiguous**

⟨array⟩ having array elements in order that are not separated by other data objects, as specified in 8.5.7

## 3.42

### **contiguous**

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

## 3.43

### **corank**

number of [codimensions](#) of a [coarray](#) (zero for objects that are not coarrays) (8.5.6)

## 3.44

### **cosubscript**

(R925) scalar integer expression in an *image-selector* (R924)

## 3.45

### **data entity**

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

## 3.46

### **data object**

#### **object**

constant (7.1.4), [variable](#) (9), or [subobject](#) of a constant (5.4.3.2.4)

## 3.47

### **decimal symbol**

character that separates the whole and fractional parts in the decimal representation of a real number in a file (13.6)

## 3.48

### **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](#).

## 3.49

### **default initialization**

mechanism for automatically initializing pointer components to have a defined pointer association status, and nonpointer components to have a particular value (7.5.4.6)

## 3.50

### **default-initialized**

⟨[subcomponent](#)⟩ subject to a [default initialization](#) specified in the type definition for that [component](#) (7.5.4.6)

## 3.51

### **definable**

capable of [definition](#) and permitted to become [defined](#)

**3.52****defined**

⟨data object⟩ has a valid value

**3.53****defined**

⟨pointer⟩ has a pointer association status of associated or [disassociated](#) (19.5.2.2)

**3.54****defined assignment**

assignment defined by a procedure (10.2.1.4, 15.4.3.4.3)

**3.55****defined input/output**

input/output defined by a procedure and accessed via a *defined-io-generic-spec* (R1509, 12.6.4.8)

**3.56****defined operation**

operation defined by a procedure (10.1.6.1, 15.4.3.4.2)

**3.57****definition**

⟨data object⟩ process by which the data object becomes defined (19.6.5)

**3.58****definition**

⟨derived type (7.5.2), enumeration (7.6), or procedure (15.6)⟩ specification of the type, enumeration, or procedure

**3.59****descendant**

⟨module or submodule⟩ submodule that extends that module or submodule or that extends another descendant thereof (14.2.3)

**3.60****designator**

name followed by zero or more component selectors, complex part selectors, array section selectors, array element selectors, image selectors, and substring selectors (9.1)

**3.60.1****complex part designator**

designator that designates the real or imaginary part of a complex [data object](#), independently of the other part (9.4.4)

**3.60.2****object designator****data object designator**

[designator](#) for a [data object](#)

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

**3.60.3****procedure designator**

[designator](#) for a procedure

**3.61****disassociated**

⟨pointer association⟩ [pointer association](#) status of not being associated with any [target](#) and not being [undefined](#) (19.5.2.2)

## 3.62

### **disassociated**

⟨[pointer](#)⟩ has a [pointer association](#) status of disassociated

## 3.63

### **dummy argument**

entity whose identifier appears in a dummy argument list (R1536) in a [FUNCTION](#), [SUBROUTINE](#), [ENTRY](#), or [statement function](#) statement, or whose name can be used as an [argument keyword](#) in a reference to an [intrinsic](#) procedure or a procedure in an [intrinsic](#) module

### 3.63.1

#### **dummy data object**

[dummy argument](#) that is a data object

### 3.63.2

#### **dummy function**

[dummy procedure](#) that is a function

## 3.64

### **effective argument**

entity that is argument-associated with a dummy argument (15.5.2.3)

## 3.65

### **effective item**

scalar object resulting from the application of the rules in 12.6.3 to an input/output list

## 3.66

### **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).

### 3.66.1

#### **elemental assignment**

assignment that operates elementally

### 3.66.2

#### **elemental operation**

operation that operates elementally

### 3.66.3

#### **elemental operator**

operator in an elemental operation

### 3.66.4

#### **elemental procedure**

elemental [intrinsic](#) procedure or procedure defined by an elemental subprogram (15.8)

### 3.66.5

#### **elemental reference**

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

### 3.66.6

#### **elemental subprogram**

subprogram with the [ELEMENTAL](#) prefix (15.8.1)

**3.67****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*

**3.68****explicit initialization**

initialization of a data object by a specification statement (8.4, 8.6.7)

**3.69****extent**

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

**3.70****external file**

file that exists in a medium external to the program (12.3)

**3.71****external unit****external input/output unit**

entity that can be [connected](#) to an [external file](#) (12.5.3, 12.5.4)

**3.72****file storage unit**

unit of storage in a [stream file](#) or an unformatted [record file](#) (12.3.5)

**3.73****final subroutine**

subroutine whose name appears in a [FINAL statement](#) (7.5.6) in a type definition, and which can be automatically invoked by the processor when an object of that type is finalized (7.5.6.2)

**3.74****finalizable**

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

**3.75****finalizable**

⟨nonpointer data entity⟩ of finalizable type

**3.76****finalization**

process of calling [final subroutines](#) when one of the events listed in 7.5.6.3 occurs

**3.77****function**

procedure that is invoked by an expression

**3.78****function result**

entity that returns the value of a function (15.6.2.2)

**3.79****generic identifier**

[lexical token](#) sequence that identifies a generic set of procedures, [intrinsic](#) operations, and/or [intrinsic](#) assignments (15.4.3.4)

## 3.80

### host instance

⟨[internal procedure](#), or [dummy procedure](#) or [procedure pointer](#) associated with an [internal procedure](#)⟩ instance of the host procedure that supplies the host environment of the [internal procedure](#) (15.6.2.4)

## 3.81

### host scoping unit

#### host

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

## 3.82

### IEEE infinity

ISO/IEC/IEEE 60559:2011 conformant infinite floating-point value

## 3.83

### IEEE NaN

ISO/IEC/IEEE 60559:2011 conformant floating-point datum that does not represent a number

## 3.84

### image

instance of a Fortran program (5.3.4)

### 3.84.1

#### active image

[image](#) that has not failed or stopped (5.3.6)

### 3.84.2

#### failed image

[image](#) that has not initiated termination but which has ceased to participate in program execution (5.3.6)

### 3.84.3

#### stopped image

[image](#) that has initiated [normal termination](#) (5.3.6)

## 3.85

### image index

integer value identifying an [image](#) within a [team](#)

## 3.86

### image control statement

statement that affects the execution ordering between [images](#) (11.6)

## 3.87

### inclusive scope

nonblock [scoping unit](#) plus every [block scoping unit](#) whose [host](#) 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](#).

## 3.88

### inherit

⟨[extended type](#)⟩ acquire entities ([components](#), [type-bound procedures](#), and [type parameters](#)) through type extension from the parent type (7.5.7.2)

## 3.89

### inquiry function

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

**3.90****interface**

⟨procedure⟩ name, procedure characteristics, dummy argument names, binding label, and generic identifiers (15.4.1)

**3.90.1****abstract interface**

set of procedure characteristics with dummy argument names (15.4.1)

**3.90.2****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 (15.4.2)

**3.90.3****generic interface**

set of procedure interfaces identified by a [generic identifier](#)

**3.90.4****implicit interface**

interface of a procedure that is not an explicit interface (15.4.2, 15.4.3.8)

**3.90.5****specific interface**

[interface](#) identified by a nongeneric name

**3.91****interface block**

[abstract interface block](#), [generic interface block](#), or [specific interface block](#) (15.4.3.2)

**3.91.1****abstract interface block**

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

**3.91.2****generic interface block**

interface block with a [generic-spec](#); collection of [interface bodies](#) and procedure statements that are to be given that generic identifier

**3.91.3****specific interface block**

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

**3.92****interoperable**

⟨Fortran entity⟩ equivalent to an entity defined by or definable by the [companion processor](#) (18.3)

**3.93****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

**3.93.1****standard intrinsic**

⟨procedure or module⟩ defined in this document (16)

## 3.93.2

### **nonstandard intrinsic**

⟨procedure or module⟩ provided by a processor but not defined in this document

## 3.94

### **internal file**

character variable that is [connected](#) to an [internal unit](#) (12.4)

## 3.95

### **internal unit**

[input/output unit](#) that is [connected](#) to an [internal file](#) (12.5.4)

## 3.96

### **ISO 10646 character**

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

## 3.97

### **keyword**

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

### 3.97.1

#### **argument keyword**

word that identifies the corresponding [dummy argument](#) in an [actual argument](#) list (15.5.2.1)

### 3.97.2

#### **component keyword**

word that identifies a [component](#) in a [structure constructor](#) (7.5.10)

### 3.97.3

#### **statement keyword**

word that is part of the syntax of a statement (5.5.2)

### 3.97.4

#### **type parameter keyword**

word that identifies a [type parameter](#) in a *type-param-spec*

## 3.98

### **lexical token**

keyword, name, literal constant other than a complex literal constant, operator, label, delimiter, comma, =, =>, :, ::, ;, or % (6.2)

## 3.99

### **line**

sequence of zero or more characters

## 3.100

### **main program**

[program unit](#) that is not a [subprogram](#), [module](#), [submodule](#), or [block data program unit](#) (14.1)

## 3.101

### **masked array assignment**

[assignment statement](#) in a [WHERE statement](#) or [WHERE construct](#) (10.2.3)

## 3.102

### **module**

[program unit](#) containing (or accessing from other modules) definitions that are to be made accessible to other [program units](#) (14.2)

**3.103****name**

identifier of a program constituent, formed according to the rules given in 6.2.2

**3.104****NaN**

Not a Number, a symbolic floating-point datum (ISO/IEC/IEEE 60559:2011)

**3.105****operand**

data value that is the subject of an operator

**3.106****operator**

*intrinsic-operator*, *defined-unary-op*, or *defined-binary-op* (R608, R1003, R1023)

**3.107****passed-object dummy argument**

dummy argument of a [type-bound procedure](#) or procedure pointer component that becomes associated with the object through which the procedure is invoked (7.5.4.5)

**3.108****pointer**

[data pointer](#) or [procedure pointer](#)

**3.108.1****data pointer**

[data entity](#) with the [POINTER](#) attribute (8.5.14)

**3.108.2****procedure pointer**

procedure with the [EXTERNAL](#) and [POINTER](#) attributes (8.5.9, 8.5.14)

**3.108.3****local procedure pointer**

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

**3.109****pointer assignment**

association of a pointer with a target, by execution of a [pointer assignment statement](#) (10.2.2) or an [intrinsic assignment statement](#) (10.2.1.2) for a derived-type object that has the pointer as a subobject

**3.110****polymorphic**

⟨data entity⟩ able to be of differing [dynamic types](#) during program execution (7.3.2.3)

**3.111****preconnected**

⟨file or [unit](#)⟩ [connected](#) at the beginning of execution of the program (12.5.5)

**3.112****procedure**

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

**3.112.1****dummy procedure**

procedure that is a [dummy argument](#) (15.2.2.3)

## 3.112.2

### **external procedure**

procedure defined by an external subprogram (R503) or by means other than Fortran (15.6.3)

## 3.112.3

### **internal procedure**

procedure defined by an internal subprogram (R512)

## 3.112.4

### **module procedure**

procedure defined by a module subprogram, or a specific procedure provided by an [intrinsic module](#) (R1408)

## 3.112.5

### **pure procedure**

procedure declared or defined to be pure (15.7)

## 3.112.6

### **type-bound procedure**

procedure that is bound to a derived type and referenced via an object of that type (7.5.5)

## 3.113

### **processor**

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

## 3.114

### **processor dependent**

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

## 3.115

### **program**

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

## 3.116

### **program unit**

[main program](#), [external subprogram](#), [module](#), [submodule](#), or [block data program unit](#) (5.2.1)

## 3.117

### **rank**

number of array dimensions of a [data entity](#) (zero for a scalar entity)

## 3.118

### **record**

sequence of values or characters in a file (12.2)

## 3.119

### **record file**

file composed of a sequence of records (12.1)

## 3.120

### **reference**

[data object](#) reference, [procedure](#) reference, or [module](#) reference

## 3.120.1

### **data object reference**

appearance of a [data object designator](#) (9.1) in a context requiring its value at that point during execution

**3.120.2****function reference**

appearance of the [procedure designator](#) for a function, or operator symbol for a [defined operation](#), in a context requiring execution of the function during expression evaluation ([15.5.3](#))

**3.120.3****module reference**

appearance of a module name in a [USE statement](#) ([14.2.2](#))

**3.120.4****procedure reference**

appearance of a [procedure designator](#), operator symbol, or assignment symbol in a context requiring execution of the procedure at that point during execution; or occurrence of defined input/output ([13.7.6](#)) or derived-type finalization ([7.5.6.2](#))

**3.121****saved**

having the [SAVE attribute](#) ([8.5.16](#))

**3.122****scalar**

[data entity](#) that can be represented by a single value of the type and that is not an array ([9.5](#))

**3.123****scoping unit**

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

**3.123.1****block scoping unit**

scoping unit of a [BLOCK construct](#)

**3.124****sequence**

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

**3.125****sequence structure**

[scalar data object](#) of a [sequence type](#) ([7.5.2.3](#))

**3.126****sequence type**

derived type with the [SEQUENCE attribute](#) ([7.5.2.3](#))

**3.126.1****character sequence type**

sequence type with no [allocatable](#) or [pointer components](#), and whose [components](#) are all default character or of another character sequence type

**3.126.2****numeric sequence type**

sequence type with no [allocatable](#) or [pointer components](#), and whose [components](#) are all default complex, default integer, default logical, default real, double precision real, or of another numeric sequence type

## 3.127

### shape

array dimensionality of a data entity, represented as a rank-one array whose size is the [rank](#) 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.

## 3.128

### simply contiguous

⟨array designator or variable⟩ satisfying the conditions specified in [9.5.4](#)

Note 1 to entry: These conditions are simple ones which make it clear that the designator or variable designates a [contiguous](#) array.

## 3.129

### size

⟨[array](#)⟩ total number of elements in the array

## 3.130

### specification expression

expression satisfying the requirements specified in [10.1.11](#), thus being suitable for use in specifications

## 3.131

### specific name

name that is not a generic name

## 3.132

### standard-conforming program

program that uses only those forms and relationships described in, and has an interpretation according to, this document

## 3.133

### statement

sequence of one or more complete or partial lines satisfying a syntax rule that ends in *-stmt* ([6.3](#))

### 3.133.1

#### 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**

### 3.133.2

#### nonexecutable statement

statement that is not an [executable statement](#)

## 3.134

### statement entity

entity whose identifier has the scope of a statement or part of a statement ([19.1](#), [19.4](#))

## 3.135

### statement label

#### label

unsigned positive number of up to five digits that refers to an individual statement ([6.2.5](#))

## 3.136

### storage sequence

contiguous sequence of [storage units](#) ([19.5.3.2](#))

**3.137****storage unit**

character storage unit, numeric storage unit, file storage unit, or unspecified storage unit (19.5.3.2)

**3.137.1****character storage unit**

unit of storage that holds a default character value (19.5.3.2)

**3.137.2****numeric storage unit**

unit of storage that holds a default real, default integer, or default logical value (19.5.3.2)

**3.137.3****unspecified storage unit**

unit of storage that holds a value that is not default character, default real, double precision real, default logical, or default complex (19.5.3.2)

**3.138****stream file**

file composed of a sequence of file storage units (12.1)

**3.139****structure**

scalar data object of derived type (7.5)

**3.139.1****structure component**

component of a structure

**3.139.2****structure constructor**

syntax (*structure-constructor*, 7.5.10) that specifies a structure value or creates such a value

**3.140****submodule**

program unit that extends a module or another submodule (14.2.3)

**3.141****subobject**

portion of data object that can be referenced, and if it is a variable defined, independently of any other portion (9.4.2)

**3.142****subprogram**

*function-subprogram* (R1529) or *subroutine-subprogram* (R1534)

**3.142.1****external subprogram**

subprogram that is not contained in a main program, module, submodule, or another subprogram

**3.142.2****internal subprogram**

subprogram that is contained in a main program or another subprogram

**3.142.3****module subprogram**

subprogram that is contained in a module or submodule but is not an internal subprogram

## 3.143

### subroutine

procedure invoked by a [CALL statement](#), by [defined assignment](#), or by some operations on derived-type entities

## 3.143.1

### atomic subroutine

[intrinsic](#) subroutine that performs an action on its ATOM argument atomically ([16.5](#))

## 3.143.2

### collective subroutine

[intrinsic](#) subroutine that performs a calculation on a [team](#) of images without requiring synchronization

## 3.144

### target

entity that is [pointer associated](#) with a [pointer](#) ([19.5.2.2](#)), entity on the right-hand-side of a [pointer assignment statement](#) ([R1033](#)), or entity with the [TARGET attribute](#) ([8.5.17](#))

## 3.145

### team

ordered set of [images](#) created by execution of a [FORM TEAM statement](#), or the initial ordered set of all [images](#)

## 3.145.1

### current team

team specified by the most recently executed [CHANGE TEAM statement](#) of a [CHANGE TEAM construct](#) that has not completed execution ([11.1.5](#)), or initial team if no [CHANGE TEAM construct](#) is being executed

## 3.145.2

### initial team

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

## 3.145.3

### parent team

⟨team except for initial team⟩ current team at time of execution of the [FORM TEAM statement](#) that created the team ([11.6.9](#))

### AC1 3.145.3bis

### sibling teams

teams created by a single set of corresponding executions of the [FORM TEAM statement](#) ([11.6.9](#)) AC1

## 3.145.4

### team number

−1 which identifies the initial team, or positive integer that identifies a team AC1 among its sibling teams AC1

## 3.146

### transformational function

[intrinsic](#) function, or function in an [intrinsic](#) module, that is neither [elemental](#) nor an [inquiry function](#)

## 3.147

### 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 ([7.1](#))

## 3.147.1

### abstract type

type with the [ABSTRACT attribute](#) ([7.5.7.1](#))

## 3.147.2

### declared type

type that a data entity is declared to have, either explicitly or implicitly ([7.3.2](#), [10.1.9](#))

**3.147.3****derived type**

type defined by a type definition (7.5) or by an [intrinsic](#) module

**3.147.4****dynamic type**

type of a data entity at a particular point during execution of a program (7.3.2.3, 10.1.9)

**3.147.5****extended type**

type with the [EXTENDS](#) attribute (7.5.7.1)

**3.147.6****extensible type**

type that may be extended using the [EXTENDS](#) clause (7.5.7.1)

**3.147.7****extension type**

⟨of one type with respect to another⟩ is the same type or is an extended type whose parent type is an extension type of the other type

**3.147.8****intrinsic type**

type defined by this document that is always accessible (7.4)

**3.147.9****numeric type**

one of the types integer, real, and complex

**3.147.10****parent type**

⟨[extended type](#)⟩ type named in the [EXTENDS](#) clause

**3.147.11****type compatible**

compatibility of the type of one entity with respect to another for purposes such as [argument association](#), [pointer association](#), and allocation (7.3.2)

**3.147.12****type parameter**

value used to parameterize a type (7.2)

**3.147.12.1****assumed type parameter**

[length type parameter](#) that assumes the type parameter value from another entity

Note 1 to entry: The other entity is

- the selector for an [associate name](#),
- the *constant-expr* for a [named constant](#) of type character, or
- the [effective argument](#) for a [dummy argument](#).

**3.147.12.2****deferred type parameter**

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

**3.147.12.3**

**kind type parameter**

type parameter whose value is required to be defaulted or given by a [constant expression](#)

**3.147.12.4**

**length type parameter**

type parameter whose value is permitted to be [assumed](#), [deferred](#), or given by a [specification expression](#)

**3.147.12.5**

**type parameter inquiry**

syntax (*type-param-inquiry*) that is used to inquire the value of a type parameter of a data object ([9.4.5](#))

**3.147.12.6**

**type parameter order**

ordering of the type parameters of a type ([7.5.3.2](#)) used for derived-type specifiers (*derived-type-spec*, [7.5.9](#))

**3.148**

**ultimate argument**

nondummy entity with which a [dummy argument](#) is associated via a chain of argument associations ([15.5.2.3](#))

**3.149**

**undefined**

[data object](#) does not have a valid value

**3.150**

**undefined**

[pointer](#) does not have a pointer association status of associated or [disassociated](#) ([19.5.2.2](#))

**3.151**

**unit**

**input/output unit**

means, specified by an *io-unit*, for referring to a file ([12.5.1](#))

**3.152**

**unlimited polymorphic**

able to have any [dynamic type](#) during program execution ([7.3.2.3](#))

**3.153**

**unsaved**

not having the [SAVE attribute](#) ([8.5.16](#))

**3.154**

**variable**

[data entity](#) that can be [defined](#) and redefined during execution of a program

**3.154.1**

**event variable**

scalar variable of type [EVENT\\_TYPE](#) ([16.10.2.10](#)) from the intrinsic module [ISO\\_FORTRAN\\_ENV](#)

**3.154.2**

**local variable**

variable in a [scoping unit](#) that is not a [dummy argument](#) 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](#)

**3.154.3**

**lock variable**

scalar variable of type [LOCK\\_TYPE](#) ([16.10.2.19](#)) from the intrinsic module [ISO\\_FORTRAN\\_ENV](#)

**3.154.4****team variable**

scalar variable of type [TEAM\\_TYPE](#) (16.10.2.32) from the intrinsic module [ISO\\_FORTRAN\\_ENV](#)

**3.155****vector subscript**

*section-subscript* that is an array (9.5.3.3.3)

**3.156****whole array**

array component or array name without further qualification (9.5.2)

## 4 Notation, conformance, and compatibility

### 4.1 Notation, symbols and abbreviated terms

#### 4.1.1 Syntax rules

- 1 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:

|           |                                                                           |
|-----------|---------------------------------------------------------------------------|
| <b>is</b> | introduces a syntactic class definition                                   |
| <b>or</b> | introduces a syntactic class alternative                                  |
| [ ]       | encloses an optional item                                                 |
| [ ] ...   | encloses an optionally repeated item<br>that may occur zero or more times |
| ■         | continues a syntax rule                                                   |

- Each syntax rule is given a unique identifying number of the form R<sub>snn</sub>, 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 1

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 1 (cont.)**

|          |
|----------|
| 4        |
| 67       |
| 1999     |
| 10243852 |

### 4.1.2 Constraints

- 1 Each constraint is given a unique identifying number of the form Csn, where s is a one- or two-digit clause number and nn is a two- or three-digit sequence number within that clause.
- 2 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.
- 3 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

- <sup>1</sup> 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.

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

#### 4.1.4 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* (R502). Expression hierarchy is described rigorously in the definition of *expr* (R1022).
- 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 R815), and in a few other cases.
- 4 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

- 1 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 1

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

## 4.2 Conformance

- 1 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.
- 2 A **processor** conforms to this document 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 constraints;
  - (3) it contains 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) it contains 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) it contains the capability to detect and report the use within a submitted **program unit** of source form or characters not permitted by Clause 6;
  - (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 Clause 19;
  - (7) it contains the capability to detect and report the use within a submitted **program unit** of a **non-standard intrinsic** procedure (including one with the same name as a **standard intrinsic** procedure but with different requirements);
  - (8) it contains the capability to detect and report the use within a submitted **program unit** of a **non-standard intrinsic** module;
  - (9) it contains 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) it contains the capability to detect and report the reason for rejecting a submitted **program**.
- 3 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.
- 4 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.
- 5 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.

- 6 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.
- 7 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.
- 8 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

- 1 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  |

### 4.3.2 New intrinsic procedures

- 1 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 2008 compatibility

- 1 Except as identified in this subclause, and except for the deleted features noted in Annex B.2, this document is an upward compatible extension to the preceding Fortran International Standard, 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.
- 2 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](#).
- 3 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.

- 4 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.
- 5 Fortran 2008 permitted a **pure** function to have a result that has a **polymorphic** allocatable **ultimate component**; this document does not permit that.
- 6 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.
- 7 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.
- 8 Fortran 2008 permitted a **polymorphic** object to be finalized within a **DO CONCURRENT construct**; this document does not permit that.
- 9 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.
- 10 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.
- AC1 11 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. AC1

#### 4.3.4 Fortran 2003 compatibility

- 1 Except as identified in this subclause, this document is an upward compatible extension to ISO/IEC 1539-1:2004 (Fortran 2003). Except as identified in this subclause, any standard-conforming Fortran 2003 program remains standard-conforming under this document.
- 2 Fortran 2003 permitted a **sequence type** to have **type parameters**; that is not permitted by this document.
- 3 Fortran 2003 specified that array constructors and structure constructors of **finalizable** type are finalized. This document specifies that these constructors are not finalized.
- 4 The form produced by the G edit descriptor for some values and some input/output rounding modes differs from that specified by Fortran 2003.
- 5 Fortran 2003 required an **explicit interface** only for a procedure that was actually referenced in the scope, not merely passed as an **actual argument**. This document requires an **explicit interface** for a procedure under the conditions listed in 15.4.2.2, regardless of whether the procedure is **referenced** in the scope.
- 6 Fortran 2003 permitted the **function result** of a **pure** function to be a **polymorphic** allocatable variable, to have a **polymorphic** allocatable **ultimate component**, or to be **finalizable** by an impure **final subroutine**. These are not permitted by this document.
- 7 Fortran 2003 permitted an **INTENT (OUT)** argument of a **pure** subroutine to be **polymorphic**; that is not permitted by this document.
- 8 Fortran 2003 interpreted assignment to an allocatable variable from a nonconformable array as **intrinsic assignment**, even when an elemental **defined assignment** was in scope; this document does not permit assignment from a nonconformable array in this context.
- 9 Fortran 2003 permitted a statement function to be of parameterized derived type; this document does not permit that.
- 10 Fortran 2003 permitted a **pure** statement function to reference a **volatile** variable, and permitted a **local variable** of a **pure** subprogram to be **volatile** (provided it was not used); this document does not permit that.

### 4.3.5 Fortran 95 compatibility

- 1 Except as identified in this subclause, this document is an upward compatible extension to ISO/IEC 1539-1:1997 (Fortran 95). Except as identified in this subclause, any standard-conforming Fortran 95 program remains standard-conforming under this document.
- 2 Fortran 95 permitted defined assignment between character strings of the same rank and different kinds. This document does not permit that if both of the different kinds are ASCII, ISO 10646, or default kind.
- 3 The following Fortran 95 features might have different interpretations in this document.
  - Earlier Fortran standards had the concept of printing, meaning that column one of formatted output had special meaning for a processor-dependent (possibly empty) set of [external files](#). This could be neither detected nor specified by a standard-specified means. The interpretation of the first column is not specified by this document.
  - This document specifies a different output format for real zero values in list-directed and namelist output.
  - If the processor distinguishes between positive and negative real zero, this document requires different returned values for [ATAN2\(Y,X\)](#) when  $X < 0$  and  $Y$  is negative real zero and for [LOG\(X\)](#) and [SQRT\(X\)](#) when  $X$  is complex with  $X\%RE < 0$  and  $X\%IM$  is negative real zero.
  - This document has fewer restrictions on [constant expressions](#) than Fortran 95; this affects whether a variable is considered to be an [automatic data object](#).
  - The form produced by the G edit descriptor with  $d$  equal to zero differs from that specified by Fortran 95 for some values.

### 4.3.6 Fortran 90 compatibility

- 1 Except for the deleted features noted in Annex [B.1](#), and except as identified in this subclause, this document is an upward compatible extension to ISO/IEC 1539:1991 (Fortran 90). Any standard-conforming Fortran 90 program that does not use one of the deleted features remains standard-conforming under this document.
- 2 The [PAD= specifier](#) in the [INQUIRE statement](#) in this document returns the value UNDEFINED if there is no connection or the connection is for unformatted input/output. Fortran 90 specified YES.
- 3 Fortran 90 specified that if the second argument to [MOD](#) or [MODULO](#) was zero, the result was processor dependent. This document specifies that the second argument shall not be zero.
- 4 Fortran 90 permitted defined assignment between character strings of the same rank and different kinds. This document does not permit that if both of the different kinds are ASCII, ISO 10646, or default kind.
- 5 The following Fortran 90 features have different interpretations in this document:
  - if the processor distinguishes between positive and negative real zero, the result value of the intrinsic function [SIGN](#) when the second argument is a negative real zero;
  - formatted output of negative real values (when the output value is zero);
  - whether an expression is a [constant expression](#) (thus whether a variable is considered to be an [automatic data object](#));
  - the G edit descriptor with  $d$  equal to zero for some values.

### 4.3.7 FORTRAN 77 compatibility

- 1 Except for the deleted features noted in Annex [B.1](#), and except as identified in this subclause, this document is an upward compatible extension to ISO 1539:1980 (FORTRAN 77). Any standard-conforming FORTRAN 77 program that does not use one of the deleted features noted in Annex [B.1](#) and that does not depend on the differences specified here remains standard-conforming under this document. This document 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 might have a different interpretation under this document, yet

remain a standard-conforming program. The following FORTRAN 77 features might have different interpretations in this document.

- FORTRAN 77 permitted a processor to supply more precision derived from a default real constant than can be represented in a default real datum when the constant is used to initialize a double precision real data object in a [DATA statement](#). This document does not permit a processor this option.
- 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. This document specifies (8.6.7) that this named variable has the [SAVE attribute](#).
- FORTRAN 77 specified 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. This document specifies (12.6.4.5.3) 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](#) or [READ](#) statement.
- 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, this document specifies how rounding of values will affect the output field form, but FORTRAN 77 did not address this issue. Therefore, the form produced for certain combinations of values and G edit descriptors might differ from that produced by some FORTRAN 77 processors.
- FORTRAN 77 did not permit a processor to distinguish between positive and negative real zero; if the processor does so distinguish, the result will differ for the intrinsic function [SIGN](#) when the second argument is negative real zero, and formatted output of negative real zero will be different.

## 4.4 Deleted and obsolescent features

### 4.4.1 General

- 1 This document protects the users' investment in existing software by including all but six of the language elements of Fortran 90 that are not processor dependent. This document identifies two categories of outmoded features. The first category, deleted features, 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 and Fortran 95, but are still frequently used.

### 4.4.2 Nature of deleted features

- 1 There are two groups of deleted features. The first group contains features for which better methods existed in FORTRAN 77; these features were not included in Fortran 95, Fortran 2003, or Fortran 2008, and are not included in this document. The second group contains features for which better methods existed in Fortran 90; these features were included in Fortran 2008, but are not included in this document.

### 4.4.3 Nature of obsolescent features

- 1 Better methods existed in Fortran 90 and Fortran 95 for each obsolescent feature. It is recommended that programmers use these better methods in new programs and convert existing code to these methods.
- 2 The obsolescent features are identified in the text of this document by a distinguishing type font (4.1.5).
- 3 A future revision of this document might delete an obsolescent feature if its use has become insignificant.

## 5 Fortran concepts

### 5.1 High level syntax

- 1 This subclause introduces the syntax associated with [program units](#) and other Fortran concepts above the construct, statement, and expression levels and illustrates their relationships.

#### NOTE 1

Constraints and other information related to the rules that do not begin with R5 appear in the appropriate clause.

|       |                              |           |                                                                                                                                                           |
|-------|------------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| R501  | <i>program</i>               | <b>is</b> | <i>program-unit</i><br>[ <i>program-unit</i> ] ...                                                                                                        |
| R502  | <i>program-unit</i>          | <b>is</b> | <i>main-program</i><br><b>or</b> <i>external-subprogram</i><br><b>or</b> <i>module</i><br><b>or</b> <i>submodule</i><br><b>or</b> <i>block-data</i>       |
| R1401 | <i>main-program</i>          | <b>is</b> | [ <i>program-stmt</i> ]<br>[ <i>specification-part</i> ]<br>[ <i>execution-part</i> ]<br>[ <i>internal-subprogram-part</i> ]<br><i>end-program-stmt</i>   |
| R503  | <i>external-subprogram</i>   | <b>is</b> | <i>function-subprogram</i><br><b>or</b> <i>subroutine-subprogram</i>                                                                                      |
| R1529 | <i>function-subprogram</i>   | <b>is</b> | <i>function-stmt</i><br>[ <i>specification-part</i> ]<br>[ <i>execution-part</i> ]<br>[ <i>internal-subprogram-part</i> ]<br><i>end-function-stmt</i>     |
| R1534 | <i>subroutine-subprogram</i> | <b>is</b> | <i>subroutine-stmt</i><br>[ <i>specification-part</i> ]<br>[ <i>execution-part</i> ]<br>[ <i>internal-subprogram-part</i> ]<br><i>end-subroutine-stmt</i> |
| R1404 | <i>module</i>                | <b>is</b> | <i>module-stmt</i><br>[ <i>specification-part</i> ]<br>[ <i>module-subprogram-part</i> ]<br><i>end-module-stmt</i>                                        |
| R1416 | <i>submodule</i>             | <b>is</b> | <i>submodule-stmt</i><br>[ <i>specification-part</i> ]<br>[ <i>module-subprogram-part</i> ]<br><i>end-submodule-stmt</i>                                  |
| R1420 | <i>block-data</i>            | <b>is</b> | <i>block-data-stmt</i><br>[ <i>specification-part</i> ]<br><i>end-block-data-stmt</i>                                                                     |

|       |                                   |    |                                                                                                                                                                                                                                                   |
|-------|-----------------------------------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| R504  | <i>specification-part</i>         | is | [ <i>use-stmt</i> ] ...<br>[ <i>import-stmt</i> ] ...<br>[ <i>implicit-part</i> ]<br>[ <i>declaration-construct</i> ] ...                                                                                                                         |
| R505  | <i>implicit-part</i>              | is | [ <i>implicit-part-stmt</i> ] ...<br><i>implicit-stmt</i>                                                                                                                                                                                         |
| R506  | <i>implicit-part-stmt</i>         | is | <i>implicit-stmt</i><br>or <i>parameter-stmt</i><br>or <i>format-stmt</i><br>or <i>entry-stmt</i>                                                                                                                                                 |
| R507  | <i>declaration-construct</i>      | is | <i>specification-construct</i><br>or <i>data-stmt</i><br>or <i>format-stmt</i><br>or <i>entry-stmt</i><br>or <i>stmt-function-stmt</i>                                                                                                            |
| R508  | <i>specification-construct</i>    | is | <i>derived-type-def</i><br>or <i>enum-def</i><br>or <i>generic-stmt</i><br>or <i>interface-block</i><br>or <i>parameter-stmt</i><br>or <i>procedure-declaration-stmt</i><br>or <i>other-specification-stmt</i><br>or <i>type-declaration-stmt</i> |
| R509  | <i>execution-part</i>             | is | <i>executable-construct</i><br>[ <i>execution-part-construct</i> ] ...                                                                                                                                                                            |
| R510  | <i>execution-part-construct</i>   | is | <i>executable-construct</i><br>or <i>format-stmt</i><br>or <i>entry-stmt</i><br>or <i>data-stmt</i>                                                                                                                                               |
| R511  | <i>internal-subprogram-part</i>   | is | <i>contains-stmt</i><br>[ <i>internal-subprogram</i> ] ...                                                                                                                                                                                        |
| R512  | <i>internal-subprogram</i>        | is | <i>function-subprogram</i><br>or <i>subroutine-subprogram</i>                                                                                                                                                                                     |
| R1407 | <i>module-subprogram-part</i>     | is | <i>contains-stmt</i><br>[ <i>module-subprogram</i> ] ...                                                                                                                                                                                          |
| R1408 | <i>module-subprogram</i>          | is | <i>function-subprogram</i><br>or <i>subroutine-subprogram</i><br>or <i>separate-module-subprogram</i>                                                                                                                                             |
| R1538 | <i>separate-module-subprogram</i> | is | <i>mp-subprogram-stmt</i><br>[ <i>specification-part</i> ]<br>[ <i>execution-part</i> ]<br>[ <i>internal-subprogram-part</i> ]<br><i>end-mp-subprogram-stmt</i>                                                                                   |
| R513  | <i>other-specification-stmt</i>   | is | <i>access-stmt</i><br>or <i>allocatable-stmt</i><br>or <i>asynchronous-stmt</i>                                                                                                                                                                   |