BRITISH STANDARD

BS ISO/IEC 1539-1:2004

Incorporating Corrigenda February 2006, February 2007 and November 2008

Information technology — Programming languages — Fortran —

Part 1: Base language

ICS 35.060

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW

BSI Documents
Document Sharing Hub
British Standards

Full document details: BS ISO/IEC 1539-1:2004

National foreword

This British Standard is the UK implementation of ISO/IEC 1539-1:2004, incorporating corrigenda February 2006, February 2007 and November 2008. It supersedes BS ISO/IEC 1539-1:1997 which is withdrawn.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 February 2006 is indicated in the text by ⟦AC1⟩ ⟨AC1⟧.

The start and finish of text introduced or altered by corrigendum is indicated in the text by tags. Text altered by ISO/IEC corrigendum February 2007 is indicated in the text by ⟦AC2⟩ ⟨AC2⟧.

The start and finish of text introduced or altered by corrigendum is indicated in the text by tags. Text altered by ISO/IEC corrigendum November 2008 is indicated in the text by ⟦AC3⟩ ⟨AC3⟧.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interfaces.

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

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

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 13 December 2004

© BSI 2010

ISBN 978 0 580 65458 9

Amendments/corrigenda issued since publication

Amd. No.	Date	Comments
16275 Corrigendum No. 1	28 April 2006	Implementation of ISO/IEC corrigendum February 2006
	31 January 2010	Implementation of ISO/IEC corrigenda February 2007 and November 2008

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

INTERNATIONAL STANDARD

ISO/IEC 1539-1

Second edition
2004-11-15

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 1539-1:2004(E)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Contents

1 Overview . . . 1
1.1 Scope . . . 1
1.2 Processor . . . 1
1.3 Inclusions . . . 1
1.4 Exclusions . . . 1
1.5 Conformance . . . 2
1.6 Compatibility . . . 3
1.6.1 Fortran 95 compatibility . . . 3
1.6.2 Fortran 90 compatibility . . . 3
1.6.3 FORTRAN 77 compatibility . . . 4
1.7 Notation used in this standard . . . 4
1.7.1 Informative notes . . . 4
1.7.2 Syntax rules . . . 4
1.7.3 Constraints . . . 5
1.7.4 Assumed syntax rules . . . 6
1.7.5 Syntax conventions and characteristics . . . 6
1.7.6 Text conventions . . . 6
1.8 Deleted and obsolescent features . . . 7
1.8.1 Nature of deleted features . . . 7
1.8.2 Nature of obsolescent features . . . 7
1.9 Normative references . . . 7

2 Fortran terms and concepts . . . 9
2.1 High level syntax . . . 9
2.2 Program unit concepts . . . 11
2.2.1 Program . . . 12
2.2.2 Main program . . . 12
2.2.3 Procedure . . . 12
2.2.4 Module . . . 13
2.3 Execution concepts . . . 13
2.3.1 Executable/nonexecutable statements . . . 13
2.3.2 Statement order . . . 13
2.3.3 The END statement . . . 14
2.3.4 Execution sequence . . . 15
2.4 Data concepts . . . 15
2.4.1 Type . . . 15
2.4.2 Data value . . . 16
2.4.3 Data entity . . . 16
2.4.4 Scalar . . . 17
2.4.5 Array . . . 18
2.4.6 Pointer . . . 18
2.4.7 Storage . . . 18
2.5 Fundamental terms . . . 18
2.5.1 Name and designator . . . 19
2.5.2 Keyword . . . 19
2.5.3 Association . . . 19
2.5.4 Declaration . . . 19

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

2.5.5 Definition . . . 19
2.5.6 Reference . . . 20
2.5.7 Intrinsic . . . 20
2.5.8 Operator . . . 20
2.5.9 Sequence . . . 20
2.5.10 Companion processors . . . 21

3 Characters, lexical tokens, and source form . . . 23
3.1 Processor character set . . . 23
3.1.1 Letters . . . 23
3.1.2 Digits . . . 24
3.1.3 Underscore . . . 24
3.1.4 Special characters . . . 24
3.1.5 Other characters . . . 24
3.2 Low-level syntax . . . 25
3.2.1 Names . . . 25
3.2.2 Constants . . . 25
3.2.3 Operators . . . 25
3.2.4 Statement labels . . . 26
3.2.5 Delimiters . . . 27
3.3 Source form . . . 27
3.3.1 Free source form . . . 27
3.3.2 Fixed source form . . . 29
3.4 Including source text . . . 30

4 Types . . . 33
4.1 The concept of type . . . 33
4.1.1 Set of values . . . 33
4.1.2 Constants . . . 33
4.1.3 Operations . . . 34
4.2 Type parameters . . . 34
4.3 Relationship of types and values to objects . . . 35
4.4 Intrinsic types . . . 35
4.4.1 Integer type . . . 36
4.4.2 Real type . . . 37
4.4.3 Complex type . . . 39
4.4.4 Character type . . . 40
4.4.5 Logical type . . . 43
4.5 Derived types . . . 44
4.5.1 Derived-type definition . . . 45
4.5.2 Derived-type parameters . . . 48
4.5.3 Components . . . 49
4.5.4 Type-bound procedures . . . 56
4.5.5 Final subroutines . . . 58
4.5.6 Type extension . . . 60
4.5.7 Derived-type values . . . 62
4.5.8 Derived-type specifier . . . 62
4.5.9 Construction of derived-type values . . . 63
4.5.10 Derived-type operations and assignment . . . 65
4.6 Enumerations and enumerators . . . 66
4.7 Construction of array values . . . 67

5 Data object declarations and specifications . . . 71
5.1 Type declaration statements . . . 71

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

5.1.1 Declaration type specifiers . . . 75
5.1.2 Attributes . . . 76
5.2 Attribute specification statements . . . 85
5.2.1 Accessibility statements . . . 86
5.2.2 ALLOCATABLE statement . . . 86
5.2.3 ASYNCHRONOUS statement . . . 86
5.2.4 BIND statement . . . 87
5.2.5 DATA statement . . . 87
5.2.6 DIMENSION statement . . . 90
5.2.7 INTENT statement . . . 90
5.2.8 OPTIONAL statement . . . 90
5.2.9 PARAMETER statement . . . 90
5.2.10 POINTER statement . . . 91
5.2.11 PROTECTED statement . . . 91
5.2.12 SAVE statement . . . 91
5.2.13 TARGET statement . . . 92
5.2.14 VALUE statement . . . 92
5.2.15 VOLATILE statement . . . 92
5.3 IMPLICIT statement . . . 92
5.4 NAMELIST statement . . . 95
5.5 Storage association of data objects . . . 95
5.5.1 EQUIVALENCE statement . . . 95
5.5.2 COMMON statement . . . 98

6 Use of data objects . . . 103
6.1 Scalars . . . 104
6.1.1 Substrings . . . 104
6.1.2 Structure components . . . 104
6.1.3 Type parameter inquiry . . . 106
6.2 Arrays . . . 106
6.2.1 Whole arrays . . . 107
6.2.2 Array elements and array sections . . . 107
6.3 Dynamic association . . . 110
6.3.1 ALLOCATE statement . . . 110
6.3.2 NULLIFY statement . . . 113
6.3.3 DEALLOCATE statement . . . 114

7 Expressions and assignment . . . 117
7.1 Expressions . . . 117
7.1.1 Form of an expression . . . 117
7.1.2 Intrinsic operations . . . 121
7.1.3 Defined operations . . . 122
7.1.4 Type, type parameters, and shape of an expression . . . 123
7.1.5 Conformability rules for elemental operations . . . 125
7.1.6 Specification expression . . . 125
7.1.7 Initialization expression . . . 126
7.1.8 Evaluation of operations . . . 128
7.2 Interpretation of operations . . . 132
7.2.1 Numeric intrinsic operations . . . 133
7.2.2 Character intrinsic operation . . . 133
7.2.3 Relational intrinsic operations . . . 134
7.2.4 Logical intrinsic operations . . . 135
7.3 Precedence of operators . . . 136
7.4 Assignment . . . 138

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

7.4.1 Assignment statement . . . 138
7.4.2 Pointer assignment . . . 142
7.4.3 Masked array assignment – WHERE . . . 145
7.4.4 FORALL . . . 148

8 Execution control . . . 155
8.1 Executable constructs containing blocks . . . 155
8.1.1 Rules governing blocks . . . 155
8.1.2 IF construct . . . 156
8.1.3 CASE construct . . . 158
8.1.4 ASSOCIATE construct . . . 160
8.1.5 SELECT TYPE construct . . . 162
8.1.6 DO construct . . . 164
8.2 Branching . . . 169
8.2.1 GO TO statement . . . 169
8.2.2 Computed GO TO statement . . . 170
8.2.3 Arithmetic IF statement . . . 170
8.3 CONTINUE statement . . . 170
8.4 STOP statement . . . 170

9 Input/output statements . . . 171
9.1 Records . . . 171
9.1.1 Formatted record . . . 171
9.1.2 Unformatted record . . . 172
9.1.3 Endfile record . . . 172
9.2 External files . . . 172
9.2.1 File existence . . . 173
9.2.2 File access . . . 173
9.2.3 File position . . . 175
9.2.4 File storage units . . . 177
9.3 Internal files . . . 177
9.4 File connection . . . 178
9.4.1 Connection modes . . . 179
9.4.2 Unit existence . . . 179
9.4.3 Connection of a file to a unit . . . 179
9.4.4 Preconnection . . . 180
9.4.5 The OPEN statement . . . 180
9.4.6 The CLOSE statement . . . 184
9.5 Data transfer statements . . . 186
9.5.1 Control information list . . . 186
9.5.2 Data transfer input/output list . . . 191
9.5.3 Execution of a data transfer input/output statement . . . 193
9.5.4 Termination of data transfer statements . . . 205
9.6 Waiting on pending data transfer . . . 205
9.6.1 WAIT statement . . . 205
9.6.2 Wait operation . . . 206
9.7 File positioning statements . . . 207
9.7.1 BACKSPACE statement . . . 207
9.7.2 ENDFILE statement . . . 208
9.7.3 REWIND statement . . . 208
9.8 FLUSH statement . . . 208
9.9 File inquiry . . . 209
9.9.1 Inquiry specifiers . . . 210
9.9.2 Restrictions on inquiry specifiers . . . 216

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

9.9.3 Inquire by output list . . . 216
9.10 Error, end-of-record, and end-of-file conditions . . . 216
9.10.1 Error conditions and the ERR= specifier . . . 217
9.10.2 End-of-file condition and the END= specifier . . . 217
9.10.3 End-of-record condition and the EOR= specifier . . . 217
9.10.4 IOSTAT= specifier . . . 218
9.10.5 IOMSG= specifier . . . 218
9.11 Restrictions on input/output statements . . . 219

10 Input/output editing . . . 221
10.1 Explicit format specification methods . . . 221
10.1.1 FORMAT statement . . . 221
10.1.2 Character format specification . . . 221
10.2 Form of a format item list . . . 222
10.2.1 Edit descriptors . . . 222
10.2.2 Fields . . . 224
10.3 Interaction between input/output list and format . . . 224
10.4 Positioning by format control . . . 225
10.5 Decimal symbol . . . 226
10.6 Data edit descriptors . . . 226
10.6.1 Numeric editing . . . 226
10.6.2 Logical editing . . . 232
10.6.3 Character editing . . . 232
10.6.4 Generalized editing . . . 233
10.6.5 User-defined derived-type editing . . . 235
10.7 Control edit descriptors . . . 235
10.7.1 Position editing . . . 235
10.7.2 Slash editing . . . 236
10.7.3 Colon editing . . . 236
10.7.4 SS, SP, and S editing . . . 237
10.7.5 P editing . . . 237
10.7.6 BN and BZ editing . . . 237
10.7.7 RU, RD, RZ, RN, RC, and RP editing . . . 238
10.7.8 DC and DP editing . . . 238
10.8 Character string edit descriptors . . . 238
10.9 List-directed formatting . . . 238
10.9.1 List-directed input . . . 239
10.9.2 List-directed output . . . 241
10.10 Namelist formatting . . . 242
10.10.1 Namelist input . . . 243
10.10.2 Namelist output . . . 246

11 Program units . . . 249
11.1 Main program . . . 249
11.2 Modules . . . 250
11.2.1 The USE statement and use association . . . 251
11.3 Block data program units . . . 253

12 Procedures . . . 255
12.1 Procedure classifications . . . 255
12.1.1 Procedure classification by reference . . . 255
12.1.2 Procedure classification by means of definition . . . 255
12.2 Characteristics of procedures . . . 256
12.2.1 Characteristics of dummy arguments . . . 256

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

12.2.2 Characteristics of function results . . . 257
12.3 Procedure interface . . . 257
12.3.1 Implicit and explicit interfaces . . . 257
12.3.2 Specification of the procedure interface . . . 258
12.4 Procedure reference . . . 266
12.4.1 Actual arguments, dummy arguments, and argument association . . . 268
12.4.2 Function reference . . . 276
12.4.3 Subroutine reference . . . 276
12.4.4 Resolving named procedure references . . . 276
12.4.5 Resolving type-bound procedure references . . . 278
12.5 Procedure definition . . . 279
12.5.1 Intrinsic procedure definition . . . 279
12.5.2 Procedures defined by subprograms . . . 279
12.5.3 Definition and invocation of procedures by means other than Fortran . . . 285
12.5.4 Statement function . . . 285
12.6 Pure procedures . . . 286
12.7 Elemental procedures . . . 287
12.7.1 Elemental procedure declaration and interface . . . 287
12.7.2 Elemental function actual arguments and results . . . 288
12.7.3 Elemental subroutine actual arguments . . . 289

13 Intrinsic procedures and modules . . . 291
13.1 Classes of intrinsic procedures . . . 291
13.2 Arguments to intrinsic procedures . . . 291
13.2.1 The shape of array arguments . . . 292
13.2.2 Mask arguments . . . 292
13.3 Bit model . . . 292
13.4 Numeric models . . . 293
13.5 Standard generic intrinsic procedures . . . 294
13.5.1 Numeric functions . . . 294
13.5.2 Mathematical functions . . . 294
13.5.3 Character functions . . . 295
13.5.4 Kind functions . . . 295
13.5.5 Miscellaneous type conversion functions . . . 295
13.5.6 Numeric inquiry functions . . . 295
13.5.7 Array inquiry functions . . . 296
13.5.8 Other inquiry functions . . . 296
13.5.9 Bit manipulation procedures . . . 296
13.5.10 Floating-point manipulation functions . . . 296
13.5.11 Vector and matrix multiply functions . . . 297
13.5.12 Array reduction functions . . . 297
13.5.13 Array construction functions . . . 297
13.5.14 Array location functions . . . 297
13.5.15 Null function . . . 297
13.5.16 Allocation transfer procedure . . . 297
13.5.17 Random number subroutines . . . 297
13.5.18 System environment procedures . . . 298
13.6 Specific names for standard intrinsic functions . . . 298
13.7 Specifications of the standard intrinsic procedures . . . 300
13.8 Standard intrinsic modules . . . 359
13.8.1 The IEEE modules . . . 359
13.8.2 The ISO_FORTRAN_ENV intrinsic module . . . 359
13.8.3 The ISO_C_BINDING module . . . 360

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

14 Exceptions and IEEE arithmetic . . . 361
14.1 Derived types and constants defined in the modules . . . 362
14.2 The exceptions . . . 363
14.3 The rounding modes . . . 365
14.4 Underflow mode . . . 365
14.5 Halting . . . 366
14.6 The floating point status . . . 366
14.7 Exceptional values . . . 366
14.8 IEEE arithmetic . . . 366
14.9 Tables of the procedures . . . 367
14.9.1 Inquiry functions . . . 367
14.9.2 Elemental functions . . . 368
14.9.3 Kind function . . . 368
14.9.4 Elemental subroutines . . . 368
14.9.5 Nonelemental subroutines . . . 368
14.10 Specifications of the procedures . . . 369
14.11 Examples . . . 384

15 Interoperability with C . . . 389
15.1 The ISO_C_BINDING intrinsic module . . . 389
15.1.1 Named constants and derived types in the module . . . 389
15.1.2 Procedures in the module . . . 390
15.2 Interoperability between Fortran and C entities . . . 393
15.2.1 Interoperability of intrinsic types . . . 394
15.2.2 Interoperability with C pointer types . . . 395
15.2.3 Interoperability of derived types and C struct types . . . 396
15.2.4 Interoperability of scalar variables . . . 397
15.2.5 Interoperability of array variables . . . 397
15.2.6 Interoperability of procedures and procedure interfaces . . . 398
15.3 Interoperation with C global variables . . . 400
15.3.1 Binding labels for common blocks and variables . . . 401
15.4 Interoperation with C functions . . . 401
15.4.1 Binding labels for procedures . . . 401
15.4.2 Exceptions and IEEE arithmetic procedures . . . 402

16 Scope, association, and definition . . . 403
16.1 Scope of global identifiers . . . 403
16.2 Scope of local identifiers . . . 404
16.2.1 Local identifiers that are the same as common block names . . . 405
16.2.2 Function results . . . 405
16.2.3 Restrictions on generic declarations . . . 405
16.2.4 Components, type parameters, and bindings . . . 406
16.2.5 Argument keywords . . . 407
16.3 Statement and construct entities . . . 407
16.4 Association . . . 408
16.4.1 Name association . . . 408
16.4.2 Pointer association . . . 412
16.4.3 Storage association . . . 413
16.4.4 Inheritance association . . . 416
16.4.5 Establishing associations . . . 416
16.5 Definition and undefinition of variables . . . 417
16.5.1 Definition of objects and subobjects . . . 417
16.5.2 Variables that are always defined . . . 417
16.5.3 Variables that are initially defined . . . 417

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

16.5.4 Variables that are initially undefined . . . 418
16.5.5 Events that cause variables to become defined . . . 418
16.5.6 Events that cause variables to become undefined . . . 419
16.5.7 Variable definition context . . . 421

Annex A (informative)Glossary of technical terms . . . 423

Annex B (informative)Decremental features . . . 435
B.1 Deleted features . . . 435
B.2 Obsolescent features . . . 436
B.2.1 Alternate return . . . 436
B.2.2 Computed GO TO statement . . . 436
B.2.3 Statement functions . . . 436
B.2.4 DATA statements among executables . . . 437
B.2.5 Assumed character length functions . . . 437
B.2.6 Fixed form source . . . 437
B.2.7 CHARACTER* form of CHARACTER declaration . . . 437

Annex C (informative)Extended notes . . . 439
C.1 Section 4 notes . . . 439
C.1.1 Intrinsic and derived types (4.4, 4.5) . . . 439
C.1.2 Selection of the approximation methods (4.4.2) . . . 440
C.1.3 Type extension and component accessibility (4.5.1.1, 4.5.3) . . . 440
C.1.4 Abstract types . . . 441
C.1.5 Pointers (4.5.1) . . . 442
C.1.6 Structure constructors and generic names . . . 443
C.1.7 Generic type-bound procedures . . . 445
C.1.8 Final subroutines (4.5.5, 4.5.5.1, 4.5.5.2, 4.5.5.3) . . . 446
C.2 Section 5 notes . . . 448
C.2.1 The POINTER attribute (5.1.2.11) . . . 448
C.2.2 The TARGET attribute (5.1.2.14) . . . 449
C.2.3 The VOLATILE attribute (5.1.2.16) . . . 449
C.3 Section 6 notes . . . 450
C.3.1 Structure components (6.1.2) . . . 450
C.3.2 Allocation with dynamic type (6.3.1) . . . 451
C.3.3 Pointer allocation and association . . . 452
C.4 Section 7 notes . . . 453
C.4.1 Character assignment . . . 453
C.4.2 Evaluation of function references . . . 453
C.4.3 Pointers in expressions . . . 453
C.4.4 Pointers on the left side of an assignment . . . 453
C.4.5 An example of a FORALL construct containing a WHERE construct . . . 454
C.4.6 Examples of FORALL statements . . . 455
C.5 Section 8 notes . . . 456
C.5.1 Loop control . . . 456
C.5.2 The CASE construct . . . 456
C.5.3 Examples of DO constructs . . . 456
C.5.4 Examples of invalid DO constructs . . . 458
C.6 Section 9 notes . . . 459
C.6.1 External files (9.2) . . . 459
C.6.2 Nonadvancing input/output (9.2.3.1) . . . 461
C.6.3 Asynchronous input/output . . . 462
C.6.4 OPEN statement (9.4.5) . . . 463
C.6.5 Connection properties (9.4.3) . . . 464

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

C.6.6 CLOSE statement (9.4.6) . . . 465
C.7 Section 10 notes . . . 465
C.7.1 Number of records (10.3, 10.4, 10.7.2) . . . 465
C.7.2 List-directed input (10.9.1) . . . 466
C.8 Section 11 notes . . . 466
C.8.1 Main program and block data program unit (11.1, 11.3) . . . 466
C.8.2 Dependent compilation (11.2) . . . 467
C.8.3 Examples of the use of modules . . . 469
C.9 Section 12 notes . . . 475
C.9.1 Portability problems with external procedures (12.3.2.2) . . . 475
C.9.2 Procedures defined by means other than Fortran (12.5.3) . . . 476
C.9.3 Procedure interfaces (12.3) . . . 476
C.9.4 Abstract interfaces (12.3) and procedure pointer components (4.5) . . . 476
C.9.5 Argument association and evaluation (12.4.1.2) . . . 478
C.9.6 Pointers and targets as arguments (12.4.1.2) . . . 479
C.9.7 Polymorphic Argument Association (12.4.1.3) . . . 480
C.10 Section 15 notes . . . 482
C.10.1 Runtime environments . . . 482
C.10.2 Examples of Interoperation between Fortran and C Functions . . . 482
C.11 Section 16 notes . . . 487
C.11.1 Examples of host association (16.4.1.3) . . . 487
C.11.2 Rules ensuring unambiguous generics (16.2.3) . . . 488
C.12 Array feature notes . . . 493
C.12.1 Summary of features . . . 493
C.12.2 Examples . . . 495
C.12.3 FORmula TRANslation and array processing . . . 499
C.12.4 Sum of squared residuals . . . 500
C.12.5 Vector norms: infinity-norm and one-norm . . . 500
C.12.6 Matrix norms: infinity-norm and one-norm . . . 500
C.12.7 Logical queries . . . 500
C.12.8 Parallel computations . . . 501
C.12.9 Example of element-by-element computation . . . 501
C.12.10 Bit manipulation and inquiry procedures . . . 502

Annex D (informative)Syntax rules . . . 503
D.1 Extract of all syntax rules . . . 503
D.2 Syntax rule cross-reference . . . 541

Annex E (informative)Index . . . 553

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

List of Tables

2.1 Requirements on statement ordering 14
2.2 Statements allowed in scoping units 14

3.1 Special characters 24

6.1 Subscript order value 108

7.1 Type of operands and results for intrinsic operators 121
7.2 Interpretation of the numeric intrinsic operators 133
7.3 Interpretation of the character intrinsic operator // 134
7.4 Interpretation of the relational intrinsic operators 134
7.5 Interpretation of the logical intrinsic operators 135
7.6 The values of operations involving logical intrinsic operators 135
7.7 Categories of operations and relative precedence 136
7.8 Type conformance for the intrinsic assignment statement 138
7.9 Numeric conversion and the assignment statement 141

13.1 Characteristics of the result of NULL ( ) 341

15.1 Names of C characters with special semantics 390
15.2 Interoperability between Fortran and C types 394

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 1539-1:2004 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

This second edition cancels and replaces the first edition (ISO/IEC 1539-1:1997), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 1539-1:1997/Cor. 1:2001 and ISO/IEC 1539-1:1997/Cor. 2:2002.

ISO/IEC 1539 consists of the following parts, under the general title *Information technology — Programming languages — Fortran*:

— *Part 1: Base language*

— *Part 2: Varying length character strings*

— *Part 3: Conditional compilation*

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Introduction

International Standard programming language Fortran

This part of the International Standard comprises the specification of the base Fortran language, informally known as Fortran 2003. With the limitations noted in 1.6.2, the syntax and semantics of Fortran 95 are contained entirely within Fortran 2003. Therefore, any standard-conforming Fortran 95 program not affected by such limitations is a standard conforming Fortran 2003 program. New features of Fortran 2003 can be compatibly incorporated into such Fortran 95 programs, with any exceptions indicated in the text of this part of the standard.

Fortran 2003 contains several extensions to Fortran 95; among them are:

(1) Derived-type enhancements: parameterized derived types (allows the kind, length, or shape of a derived type's components to be chosen when the derived type is used), mixed component accessibility (allows different components to have different accessibility), public entities of private type, improved structure constructors, and finalizers.

(2) Object oriented programming support: enhanced data abstraction (allows one type to extend the definition of another type), polymorphism (allows the type of a variable to vary at runtime), dynamic type allocation, SELECT TYPE construct (allows a choice of execution flow depending upon the type a polymorphic object currently has), and type-bound procedures.

(3) The ASSOCIATE construct (allows a complex expression or object to be denoted by a simple symbol).

(4) Data manipulation enhancements: allocatable components, deferred type parameters, VOLATILE attribute, explicit type specification in array constructors, INTENT specification of pointer arguments, specified lower bounds of pointer assignment and pointer rank remapping, extended initialization expressions, MAX and MIN intrinsics for character type, and enhanced complex constants.

(5) Input/output enhancements: asynchronous transfer operations (allows a program to continue to process data while an input/output transfer occurs), stream access (allows access to a file without reference to any record structure), user specified transfer operations for derived types, user specified control of rounding during format conversions, the FLUSH statement, named constants for preconnected units, regularization of input/output keywords, and access to input/output error messages.

(6) Procedure pointers.

(7) Scoping enhancements: the ability to rename defined operators (supports greater data abstraction) and control of host association into interface bodies.

(8) Support for IEC 60559 (IEEE 754) exceptions and arithmetic (to the extent a processor's arithmetic supports the IEC International Standard).

(9) Interoperability with the C programming language (allows portable access to many libraries and the low-level facilities provided by C and allows the portable use of Fortran libraries by programs written in C).

(10) Support for international usage: (ISO 10646) and choice of decimal or comma in numeric formatted input/output.

(11) Enhanced integration with the host operating system: access to command line arguments, environment variables, and the processor's error messages.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

Organization of this part of ISO/IEC 1539

This part of ISO/IEC 1539 is organized in 16 sections, dealing with 8 conceptual areas. These 8 areas, and the sections in which they are treated, are:

High/low level concepts	Sections 1, 2, 3
Data concepts	Sections 4, 5, 6
Computations	Sections 7, 13, 14
Execution control	Section 8
Input/output	Sections 9, 10
Program units	Sections 11, 12
Interoperability with C	Section 15
Scoping and association rules	Section 16

It also contains the following nonnormative material:

Glossary	A
Decremental features	B
Extended notes	C
Syntax rules	D
Index	E

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

blank

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Information technology — Programming languages — Fortran —

Part 1: Base Language

Section 1: Overview

1.1 Scope

ISO/IEC 1539 is a multipart International Standard; the parts are published separately. This publication, ISO/IEC 1539-1, which is the first part, specifies the form and establishes the interpretation of programs expressed in the base Fortran language. The purpose of this part of ISO/IEC 1539 is to promote portability, reliability, maintainability, and efficient execution of Fortran programs for use on a variety of computing systems. The second part, ISO/IEC 1539-2, defines additional facilities for the manipulation of character strings of variable length. The third part, ISO/IEC 1539-3, defines a standard conditional compilation facility for Fortran. A processor conforming to part 1 need not conform to ISO/IEC 1539-2 or ISO/IEC 1539-3; however, conformance to either assumes conformance to this part. Throughout this publication, the term "this standard" refers to ISO/IEC 1539-1.

1.2 Processor

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

1.3 Inclusions

This standard specifies

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

1.4 Exclusions

This standard does not specify

(1) The mechanism by which programs are transformed for use on computing systems,
(2) The operations required for setup and control of the use of programs on computing systems,
(3) The method of transcription of programs or their input or output data to or from a storage medium,
(4) The program and processor behavior when this standard fails to establish an interpretation except for the processor detection and reporting requirements in items (2) through (8) of 1.5,

AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

(5) 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,
(6) The physical properties of the representation of quantities and the method of rounding, approximating, or computing numeric values on a particular processor,
(7) The physical properties of input/output records, files, and units, or
(8) The physical properties and implementation of storage.

1.5 Conformance

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

A processor conforms to this standard if

(1) It executes any standard-conforming program in a manner that fulfills the interpretations herein, subject to any limits that the processor may impose on the size and complexity of the program;
(2) It contains the capability to detect and report the use within a submitted program unit of a form designated herein as obsolescent, insofar as such use can be detected by reference to the numbered syntax rules and constraints;
(3) It contains the capability to detect and report the use within a submitted program unit of an additional form or relationship that is not permitted by the numbered syntax rules or 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 (4.4);
(5) It contains the capability to detect and report the use within a submitted program unit of source form or characters not permitted by Section 3;
(6) It contains the capability to detect and report the use within a submitted program of name usage not consistent with the scope rules for names, labels, operators, and assignment symbols in Section 16;
(7) It contains the capability to detect and report the use within a submitted program unit of intrinsic procedures whose names are not defined in Section 13; and
(8) It contains the capability to detect and report the reason for rejecting a submitted program.

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

A standard-conforming processor may allow additional forms and relationships provided that such additions do not conflict with the standard forms and relationships. However, a standard-conforming processor may allow additional intrinsic procedures even though this could cause a conflict with the name of a procedure in a standard-conforming program. If such a conflict occurs and involves the name of an external procedure, the processor is permitted to use the intrinsic procedure unless the name is given the EXTERNAL attribute (5.1.2.6) in the scoping unit (16). A standard-conforming program shall not use nonstandard intrinsic procedures or modules that have been added by the processor.

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

In some cases, this standard allows the provision of facilities that are not completely specified in the

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

standard. These facilities are identified as **processor dependent**. They shall be provided, with methods or semantics determined by the processor.

NOTE 1.1

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

The processor should be accompanied by documentation that specifies the methods or semantics of processor-dependent facilities.

1.6 Compatibility

Each Fortran International Standard since ISO 1539:1980 (informally referred to as FORTRAN 77), defines more intrinsic procedures than the previous one. Therefore, a Fortran program conforming to an older standard may 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.

1.6.1 Fortran 95 compatibility

Except as identified in this section, this standard is an upward compatible extension to the preceding Fortran International Standard, ISO/IEC 1539-1:1997 (Fortran 95). Any standard-conforming Fortran 95 program remains standard-conforming under this standard. The following Fortran 95 features may have different interpretations in this standard:

(1) 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 standard.

(2) This standard specifies a different output format for real zero values in list-directed and namelist output.

(3) If the processor can distinguish between positive and negative real zero, this standard 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 REAL(X) $<$ 0 and negative zero imaginary part.

1.6.2 Fortran 90 compatibility

Except for the deleted features noted in Annex B.1, and except as identified in this section, this standard 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 standard.

The PAD= specifier in the INQUIRE statement in this standard returns the value UNDEFINED if there is no connection or the connection is for unformatted input/output. Fortran 90 specified YES.

Fortran 90 specified that if the second argument to MOD or MODULO was zero, the result was processor dependent. This standard specifies that the second argument shall not be zero.

AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

1.6.3 Fortran 77 compatibility

Except for the deleted features noted in Annex B.1, and except as identified in this section, this standard 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 standard. This standard restricts the behavior for some features that were processor dependent in Fortran 77. Therefore, a standard-conforming Fortran 77 program that uses one of these processor-dependent features may have a different interpretation under this standard, yet remain a standard-conforming program. The following Fortran 77 features may have different interpretations in this standard:

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

(2) If a named variable that was not in a common block was initialized in a DATA statement and did not have the SAVE attribute specified, Fortran 77 left its SAVE attribute processor dependent. This standard specifies (5.2.5) that this named variable has the SAVE attribute.

(3) 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 standard specifies (9.5.3.4.2) that the input record is logically padded with blanks if there are not enough characters in the record, unless the PAD= specifier with the value 'NO' is specified in an appropriate OPEN or READ statement.

(4) 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 standard specifies how rounding of values will affect the output field form, but Fortran 77 did not address this issue. Therefore, some Fortran 77 processors may produce an output form different from the output form produced by Fortran 2003 processors for certain combinations of values and G edit descriptors.

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

1.7 Notation used in this standard

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

1.7.1 Informative notes

Informative notes of explanation, rationale, examples, and other material are interspersed with the normative body of this publication. The informative material is nonnormative; it is identified by being in shaded, framed boxes that have numbered headings beginning with "NOTE."

1.7.2 Syntax rules

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

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

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

(2) 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:

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

(3) The syntactic metasymbols used are:

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

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

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

NOTE 1.2

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

If particular entities are substituted for *digit*, actual digit strings might be:

4
67
1999
10243852

1.7.3 Constraints

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

AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

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

Some constraints are not associated with particular syntax rules. The effect of such a constraint is similar to that of a restriction stated in the text, except that a processor is required to have the capability to detect and report violations of constraints (1.5). 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.

1.7.4 Assumed syntax rules

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

R101	*xyz-list*	**is**	*xyz* [, *xyz*] ...
R102	*xyz-name*	**is**	*name*
R103	*scalar-xyz*	**is**	*xyz*

C101 (R103) *scalar-xyz* shall be scalar.

1.7.5 Syntax conventions and characteristics

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

(2) The rules on statement ordering are described rigorously in the definition of *program-unit* (R202). Expression hierarchy is described rigorously in the definition of *expr* (R722).

(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 R510), 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*" (R404) and "*length-selector*" (R425).

(5) The term "*subscript*" (for example, R618, R619, and R620) is used consistently in array definitions.

1.7.6 Text conventions

In the 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". Boldface words are used in the text where they are first defined with a specialized meaning. The descriptions of obsolescent features appear in a smaller type size.

NOTE 1.3

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

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

1.8 Deleted and obsolescent features

This standard protects the users' investment in existing software by including all but five of the language elements of Fortran 90 that are not processor dependent. This standard identifies two categories of outmoded features. There are five in the first category, **deleted features**, which consists of features considered to have been redundant in FORTRAN 77 and largely unused in Fortran 90. Those in the second category, **obsolescent features**, are considered to have been redundant in Fortran 90 and Fortran 95, but are still frequently used.

1.8.1 Nature of deleted features

(1) Better methods existed in FORTRAN 77.

(2) These features are not included in Fortran 95 or this revision of Fortran.

1.8.2 Nature of obsolescent features

(1) Better methods existed in Fortran 90 and Fortran 95.

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

(3) These features are identified in the text of this document by a distinguishing type font (1.7.6).

(4) If the use of these features becomes insignificant, future Fortran standards committees should consider deleting them.

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

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

1.9 Normative references

The following referenced standards are indispensable for the application of this standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced standard (including any amendments) applies.

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

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

ISO/IEC 9899:1999, *Information technology—Programming languages—C.*

ISO/IEC 10646-1:2000, *Information technology—Universal multiple-octet coded character set (UCS)—Part 1: Architecture and basic multilingual plane.*

IEC 60559 (1989-01), *Binary floating-point arithmetic for microprocessor systems.*

ISO/IEC 646:1991 (International Reference Version) is the international equivalent of ANSI X3.4-1986, commonly known as ASCII.

This standard refers to ISO/IEC 9899:1999 as the C International Standard.

Because IEC 60559 (1989-01) was originally IEEE 754-1985, *Standard for binary floating-point arithmetic*, and is widely known by this name, this standard refers to it as the IEEE International Standard.

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

(Blank page)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Section 2: Fortran terms and concepts

2.1 High level syntax

This section introduces the terms associated with program units and other Fortran concepts above the construct, statement, and expression levels and illustrates their relationships. The notation used in this standard is described in 1.7.

NOTE 2.1

Constraints and other information related to the rules that do not begin with R2 appear in the appropriate section.

R201 *program* **is** *program-unit*
[*program-unit*] ...

A *program* shall contain exactly one *main-program program-unit* or a main program defined by means other than Fortran, but not both.

R202 *program-unit* **is** *main-program*
or *external-subprogram*
or *module*
or *block-data*

R1101 *main-program* **is** [*program-stmt*]
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-program-stmt

R203 *external-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*

R1223 *function-subprogram* **is** *function-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-function-stmt

R1231 *subroutine-subprogram* **is** *subroutine-stmt*
[*specification-part*]
[*execution-part*]
[*internal-subprogram-part*]
end-subroutine-stmt

R1104 *module* **is** *module-stmt*
[*specification-part*]
[*module-subprogram-part*]
end-module-stmt

R1116 *block-data* **is** *block-data-stmt*
[*specification-part*]
end-block-data-stmt

R204 *specification-part* **is** [*use-stmt*] ...
[*import-stmt*] ...
[*implicit-part*]
[*declaration-construct*] ...

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

R205	*implicit-part*	**is**	[*implicit-part-stmt*] ...
			implicit-stmt
R206	*implicit-part-stmt*	**is**	*implicit-stmt*
		or	*parameter-stmt*
		or	*format-stmt*
		or	*entry-stmt*
R207	*declaration-construct*	**is**	*derived-type-def*
		or	*entry-stmt*
		or	*enum-def*
		or	*format-stmt*
		or	*interface-block*
		or	*parameter-stmt*
		or	*procedure-declaration-stmt*
		or	*specification-stmt*
		or	*type-declaration-stmt*
		or	*stmt-function-stmt*
R208	*execution-part*	**is**	*executable-construct*
			[*execution-part-construct*] ...
R209	*execution-part-construct*	**is**	*executable-construct*
		or	*format-stmt*
		or	*entry-stmt*
		or	*data-stmt*
R210	*internal-subprogram-part*	**is**	*contains-stmt*
			internal-subprogram
			[*internal-subprogram*] ...
R211	*internal-subprogram*	**is**	*function-subprogram*
		or	*subroutine-subprogram*
R1107	*module-subprogram-part*	**is**	*contains-stmt*
			module-subprogram
			[*module-subprogram*] ...
R1108	*module-subprogram*	**is**	*function-subprogram*
		or	*subroutine-subprogram*
R212	*specification-stmt*	**is**	*access-stmt*
		or	*allocatable-stmt*
		or	*asynchronous-stmt*
		or	*bind-stmt*
		or	*common-stmt*
		or	*data-stmt*
		or	*dimension-stmt*
		or	*equivalence-stmt*
		or	*external-stmt*
		or	*intent-stmt*
		or	*intrinsic-stmt*
		or	*namelist-stmt*
		or	*optional-stmt*
		or	*pointer-stmt*
		or	*protected-stmt*
		or	*save-stmt*
		or	*target-stmt*
		or	*volatile-stmt*
		or	*value-stmt*
R213	*executable-construct*	**is**	*action-stmt*
		or	*associate-construct*
		or	*case-construct*

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

		or	*do-construct*
		or	*forall-construct*
		or	*if-construct*
		or	*select-type-construct*
		or	*where-construct*
R214	*action-stmt*	**is**	*allocate-stmt*
		or	*assignment-stmt*
		or	*backspace-stmt*
		or	*call-stmt*
		or	*close-stmt*
		or	*continue-stmt*
		or	*cycle-stmt*
		or	*deallocate-stmt*
		or	*endfile-stmt*
		or	*end-function-stmt*
		or	*end-program-stmt*
		or	*end-subroutine-stmt*
		or	*exit-stmt*
		or	*flush-stmt*
		or	*forall-stmt*
		or	*goto-stmt*
		or	*if-stmt*
		or	*inquire-stmt*
		or	*nullify-stmt*
		or	*open-stmt*
		or	*pointer-assignment-stmt*
		or	*print-stmt*
		or	*read-stmt*
		or	*return-stmt*
		or	*rewind-stmt*
		or	*stop-stmt*
		or	*wait-stmt*
		or	*where-stmt*
		or	*write-stmt*
		or	*arithmetic-if-stmt*
		or	*computed-goto-stmt*

C201 (R208) An *execution-part* shall not contain an *end-function-stmt*, *end-program-stmt*, or *end-subroutine-stmt*.

2.2 Program unit concepts

Program units are the fundamental components of a Fortran program. A **program unit** may be a main program, an external subprogram, a module, or a block data program unit. A subprogram may be a function subprogram or a subroutine subprogram. A module contains definitions that are to be made accessible to other program units. A block data program unit is used to specify initial values for data objects in named common blocks. Each type of program unit is described in Section 11 or 12. An **external subprogram** is a subprogram that is not in a main program, a module, or another subprogram. An **internal subprogram** is a subprogram that is in a main program or another subprogram. A **module subprogram** is a subprogram that is in a module but is not an internal subprogram.

A program unit consists of a set of nonoverlapping scoping units. A **scoping unit** is

AVA Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

(1) A program unit or subprogram, excluding any scoping units in it,
(2) A derived-type definition (4.5.1), or
(3) An interface body, excluding any scoping units in it.

A scoping unit that immediately surrounds another scoping unit is called the **host scoping unit** (often abbreviated to **host**).

2.2.1 Program

A **program** consists of exactly one main program, any number (including zero) of other kinds of program units, and any number (including zero) of external procedures and other entities defined by means other than Fortran.

NOTE 2.2

There is a restriction that there shall be no more than one unnamed block data program unit (11.3).

This standard places no ordering requirement on the program units that constitute a program, but because the public portions of a module are required to be available by the time a module reference (11.2.1) is processed, a processor may require a particular order of processing of the program units.

2.2.2 Main program

The Fortran main program is described in 11.1.

2.2.3 Procedure

A **procedure** encapsulates an arbitrary sequence of actions that may be invoked directly during program execution. Procedures are either functions or subroutines. A **function** is a procedure that is invoked in an expression; its invocation causes a value to be computed which is then used in evaluating the expression. The variable that returns the value of a function is called the **result variable**. A **subroutine** is a procedure that is invoked in a CALL statement, by a defined assignment statement, or by some operations on derived-type entities. Unless it is a pure procedure, a subroutine may be used to change the program state by changing the values of any of the data objects accessible to the subroutine; unless it is a pure procedure, a function may do this in addition to computing the function value.

Procedures are described further in Section 12.

2.2.3.1 External procedure

An **external procedure** is a procedure that is defined by an external subprogram or by means other than Fortran. An external procedure may be invoked by the main program or by any procedure of a program.

2.2.3.2 Module procedure

A **module procedure** is a procedure that is defined by a module subprogram (R1108). The module containing the subprogram is the **host** scoping unit of the module procedure.

2.2.3.3 Internal procedure

An **internal procedure** is a procedure that is defined by an internal subprogram (R211). The containing main program or subprogram is the **host** scoping unit of the internal procedure. An internal procedure is local to its host in the sense that the internal procedure is accessible within the host scoping unit and all its other internal procedures but is not accessible elsewhere.

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

2.2.3.4 Interface block

An **interface body** describes an abstract interface or the interface of a dummy procedure, external procedure, procedure pointer, or type-bound procedure.

An **interface block** is a specific interface block, an abstract interface block, or a generic interface block. A specific interface block is a collection of interface bodies. A generic interface block may also be used to specify that procedures may be invoked

(1) By using a generic name,
(2) By using a defined operator,
(3) By using a defined assignment, or
(4) For derived-type input/output.

2.2.4 Module

A **module** contains (or accesses from other modules) definitions that are to be made accessible to other program units. These definitions include data object declarations, type definitions, procedure definitions, and interface blocks. A scoping unit in another program unit may access the definitions in a module. Modules are further described in Section 11.

2.3 Execution concepts

Each Fortran statement is classified as either an executable statement or a nonexecutable statement. There are restrictions on the order in which statements may appear in a program unit, and not all executable statements may appear in all contexts.

2.3.1 Executable/nonexecutable statements

Program execution is a sequence, in time, of actions. An **executable statement** is an instruction to perform or control one or more of these actions. Thus, the executable statements of a program unit determine the behavior of the program unit. The executable statements are all of those that make up the syntactic class *executable-construct*.

Nonexecutable statements do not specify actions; they are used to configure the program environment in which actions take place. The nonexecutable statements are all those not classified as executable.

2.3.2 Statement order

The syntax rules of subclause 2.1 specify the statement order within program units and subprograms. These rules are illustrated in Table 2.1 and Table 2.2. Table 2.1 shows the ordering rules for statements and applies to all program units, subprograms, and interface bodies. Vertical lines delineate varieties of statements that may be interspersed and horizontal lines delineate varieties of statements that shall not be interspersed. Internal or module subprograms shall follow a CONTAINS statement. Between USE and CONTAINS statements in a subprogram, nonexecutable statements generally precede executable statements, although the ENTRY statement, FORMAT statement, and DATA statement may appear among the executable statements. Table 2.2 shows which statements are allowed in a scoping unit.

Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

Table 2.1: **Requirements on statement ordering**

<table>
<tr><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE,
MODULE, or BLOCK DATA statement</td></tr>
<tr><td colspan="3">USE statements</td></tr>
<tr><td colspan="3">IMPORT statements</td></tr>
<tr><td rowspan="4">FORMAT
and
ENTRY
statements</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMETER
statements</td><td>IMPLICIT
statements</td></tr>
<tr><td>PARAMETER
and DATA
statements</td><td>Derived-type definitions,
interface blocks,
type declaration statements,
enumeration definitions,
procedure declarations,
specification statements,
and statement function statements</td></tr>
<tr><td>DATA
statements</td><td>Executable
constructs</td></tr>
<tr><td colspan="3">CONTAINS statement</td></tr>
<tr><td colspan="3">Internal subprograms
or module subprograms</td></tr>
<tr><td colspan="3">END statement</td></tr>
</table>

Table 2.2: **Statements allowed in scoping units**

Kind of scoping unit:	Main program	Module	Block data	External subprog	Module subprog	Internal subprog	Interface body
USE statement	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IMPORT statement	No	No	No	No	No	No	Yes
ENTRY statement	No	No	No	Yes	Yes	No	No
FORMAT statement	Yes	No	No	Yes	Yes	Yes	No
Misc. decls (see note)	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DATA statement	Yes	Yes	Yes	Yes	Yes	Yes	No
Derived-type definition	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Interface block	Yes	Yes	No	Yes	Yes	Yes	Yes
Executable statement	Yes	No	No	Yes	Yes	Yes	No
CONTAINS statement	Yes	Yes	No	Yes	Yes	No	No
Statement function statement	Yes	No	No	Yes	Yes	Yes	No

Notes for Table 2.2:

1) Misc. declarations are PARAMETER statements, IMPLICIT statements, type declaration statements, enumeration definitions, procedure declaration statements, and specification statements.

2) The scoping unit of a module does not include any module subprograms that the module contains.

2.3.3 The END statement

An *end-program-stmt*, *end-function-stmt*, *end-subroutine-stmt*, *end-module-stmt*, or *end-block-data-stmt* is an **END statement**. Each program unit, module subprogram, and internal subprogram shall have exactly one END statement. The *end-program-stmt*, *end-function-stmt*, and *end-subroutine-stmt* statements are executable, and may be branch target statements (8.2). Executing an *end-program-stmt* causes normal termination of execution of the program. Executing an *end-function-stmt* or *end-subroutine-stmt*

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

is equivalent to executing a *return-stmt* with no *scalar-int-expr*.

The *end-module-stmt* and *end-block-data-stmt* statements are nonexecutable.

2.3.4 Execution sequence

If a program contains a Fortran main program, execution of the program begins with the first executable construct of the main program. The execution of a main program or subprogram involves execution of the executable constructs within its scoping unit. When a procedure is invoked, execution begins with the first executable construct appearing after the invoked entry point. With the following exceptions, the effect of execution is as if the executable constructs are executed in the order in which they appear in the main program or subprogram until a STOP, RETURN, or END statement is executed. The exceptions are the following:

(1) Execution of a branching statement (8.2) changes the execution sequence. These statements explicitly specify a new starting place for the execution sequence.

(2) CASE constructs, DO constructs, IF constructs, and SELECT TYPE constructs contain an internal statement structure and execution of these constructs involves implicit internal branching. See Section 8 for the detailed semantics of each of these constructs.

(3) END=, ERR=, and EOR= specifiers may result in a branch.

(4) Alternate returns may result in a branch.

Internal subprograms may precede the END statement of a main program or a subprogram. The execution sequence excludes all such definitions.

Normal termination of execution of the program occurs if a STOP statement or *end-program-stmt* is executed. Normal termination of execution of a program also may occur during execution of a procedure defined by a companion processor (C International Standard 5.1.2.2.3 and 7.20.4.3). If normal termination of execution occurs within a Fortran program unit and the program incorporates procedures defined by a companion processor, the process of execution termination shall include the effect of executing the C exit() function (C International Standard 7.20.4.3).

2.4 Data concepts

Nonexecutable statements are used to specify the characteristics of the data environment. This includes typing variables, declaring arrays, and defining new types.

2.4.1 Type

A **type** is a named category of data that is characterized by a set of values, a syntax for denoting these values, and a set of operations that interpret and manipulate the values. This central concept is described in 4.1.

A type may be parameterized, in which case the set of data values, the syntax for denoting them, and the set of operations depend on the values of one or more parameters. Such a parameter is called a **type parameter** (4.2).

There are two categories of types: intrinsic types and derived types.

2.4.1.1 Intrinsic type

An **intrinsic type** is a type that is defined by the language, along with operations, and is always accessible. The intrinsic types are integer, real, complex, character, and logical. The properties of intrinsic types are described in 4.4. The intrinsic type parameters are KIND and LEN.

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

The **kind type parameter** indicates the decimal exponent range for the integer type (4.4.1), the decimal precision and exponent range for the real and complex types (4.4.2, 4.4.3), and the representation methods for the character and logical types (4.4.4, 4.4.5). The **character length parameter** specifies the number of characters for the character type.

2.4.1.2 Derived type

A **derived type** is a type that is not defined by the language but requires a type definition to declare its components. A scalar object of such a derived type is called a **structure** (5.1.1.1). Derived types may be parameterized. Assignment of structures is defined intrinsically (7.4.1.3), but there are no intrinsic operations for structures. For each derived type, a structure constructor is available to provide values (4.5.9). In addition, data objects of derived type may be used as procedure arguments and function results, and may appear in input/output lists. If additional operations are needed for a derived type, they shall be supplied as procedure definitions.

Derived types are described further in 4.5.

2.4.2 Data value

Each intrinsic type has associated with it a set of values that a datum of that type may take, depending on the values of the type parameters. The values for each intrinsic type are described in 4.4. The values that objects of a derived type may assume are determined by the type definition, type parameter values, and the sets of values of its components.

2.4.3 Data entity

A **data entity** is a data object, the result of the evaluation of an expression, or the result of the execution of a function reference (called the function result). A data entity has a type and type parameters; it may have a data value (an exception is an undefined variable). Every data entity has a rank and is thus either a scalar or an array.

2.4.3.1 Data object

A **data object** (often abbreviated to **object**) is a constant (4.1.2), a variable (6), or a subobject of a constant. The type and type parameters of a named data object may be specified explicitly (5.1) or implicitly (5.3).

Subobjects are portions of certain objects that may be referenced and defined (variables only) independently of the other portions. These include portions of arrays (array elements and array sections), portions of character strings (substrings), portions of complex objects (real and imaginary parts), and portions of structures (components). Subobjects are themselves data objects, but subobjects are referenced only by object designators or intrinsic functions. A subobject of a variable is a variable. Subobjects are described in Section 6.

Objects referenced by a name are:

a named scalar	(a scalar object)
a named array	(an array object)

Subobjects referenced by an object designator are:

an array element	(a scalar subobject)
an array section	(an array subobject)
a structure component	(a scalar or an array subobject)
a substring	(a scalar subobject)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Subobjects of complex objects may also be referenced by intrinsic functions.

2.4.3.1.1 Variable

A **variable** may have a value and may be defined and redefined during execution of a program.

A named **local variable** of the scoping unit of a module, main program, or subprogram, is a named variable that is a local entity of the scoping unit, is not a dummy argument, is not in COMMON, does not have the BIND attribute, and is not accessed by use or host association. A subobject of a named local variable is also a local variable.

2.4.3.1.2 Constant

A **constant** has a value and cannot become defined, redefined, or undefined during execution of a program. A constant with a name is called a **named constant** and has the PARAMETER attribute (5.1.2.10). A constant without a name is called a **literal constant** (4.4).

2.4.3.1.3 Subobject of a constant

A **subobject of a constant** is a portion of a constant. The portion referenced may depend on the value of a variable.

NOTE 2.3

For example, given:

```
CHARACTER (LEN = 10), PARAMETER :: DIGITS = '0123456789'
CHARACTER (LEN = 1)             :: DIGIT
INTEGER :: I
   ...
DIGIT = DIGITS (I:I)
```

DIGITS is a named constant and DIGITS (I:I) designates a subobject of the constant DIGITS.

2.4.3.2 Expression

An **expression** (7.1) produces a data entity when evaluated. An expression represents either a data reference or a computation; it is formed from operands, operators, and parentheses. The type, type parameters, value, and rank of an expression result are determined by the rules in Section 7.

2.4.3.3 Function reference

A **function reference** (12.4.2) produces a data entity when the function is executed during expression evaluation. The type, type parameters, and rank of a function result are determined by the interface of the function (12.2.2). The value of a function result is determined by execution of the function.

2.4.4 Scalar

A **scalar** is a datum that is not an array. Scalars may be of any intrinsic type or derived type.

NOTE 2.4

A structure is scalar even if it has arrays as components.

The **rank** of a scalar is zero. The shape of a scalar is represented by a rank-one array of size zero.

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

2.4.5 Array

An **array** is a set of scalar data, all of the same type and type parameters, whose individual elements are arranged in a rectangular pattern. An **array element** is one of the individual elements in the array and is a scalar. An **array section** is a subset of the elements of an array and is itself an array.

An array may have up to seven dimensions, and any **extent** (number of elements) in any dimension. The **rank** of the array is the number of dimensions; its **size** is the total number of elements, which is equal to the product of the extents. An array may have zero size. The **shape** of an array is determined by its rank and its extent in each dimension, and may be represented as a rank-one array whose elements are the extents. All named arrays shall be declared, and the rank of a named array is specified in its declaration. The rank of a named array, once declared, is constant; the extents may be constant or may vary during execution.

Two arrays are **conformable** if they have the same shape. A scalar is conformable with any array. Any intrinsic operation defined for scalar objects may be applied to conformable objects. Such operations are performed element-by-element to produce a resultant array conformable with the array operands. Element-by-element operation means corresponding elements of the operand arrays are involved in a scalar operation to produce the corresponding element in the result array, and all such element operations may be performed in any order or simultaneously. Such an operation is described as **elemental**.

A rank-one array may be constructed from scalars and other arrays and may be reshaped into any allowable array shape (4.7).

Arrays may be of any intrinsic type or derived type and are described further in 6.2.

2.4.6 Pointer

A **data pointer** is a data entity that has the POINTER attribute. A **procedure pointer** is a procedure entity that has the POINTER attribute. A **pointer** is either a data pointer or a procedure pointer.

A pointer is **associated** with a **target** by pointer assignment (7.4.2). A data pointer may also be associated with a target by allocation (6.3.1). A pointer is **disassociated** following execution of a NULLIFY statement, following pointer assignment with a disassociated pointer, by default initialization, or by explicit initialization. A data pointer may also be disassociated by execution of a DEALLOCATE statement. A disassociated pointer is not associated with a target (16.4.2).

A pointer that is not associated shall not be referenced or defined.

If a data pointer is an array, the rank is declared, but the extents are determined when the pointer is associated with a target.

2.4.7 Storage

Many of the facilities of this standard make no assumptions about the physical storage characteristics of data objects. However, program units that include storage association dependent features shall observe the storage restrictions described in 16.4.3.

2.5 Fundamental terms

For the purposes of this standard, the definitions of terms in this subclause apply.

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

2.5.1 Name and designator

A **name** is used to identify a program constituent, such as a program unit, named variable, named constant, dummy argument, or derived type. The rules governing the construction of names are given in 3.2.1. A **designator** is a name followed by zero or more component selectors, array section selectors, array element selectors, and substring selectors.

An **object designator** is a designator for a data object. A **procedure designator** is a designator for a procedure.

NOTE 2.5

An object name is a special case of an object designator.

2.5.2 Keyword

The term **keyword** is used in two ways.

(1) It is used to describe a word that is part of the syntax of a statement. These keywords are not reserved words; that is, names with the same spellings are allowed. In the syntax rules, such keywords appear literally. In descriptive text, this meaning is denoted by the term "keyword" without any modifier. Examples of statement keywords are: IF, READ, UNIT, KIND, and INTEGER.

(2) It is used to denote names that identify items in a list. In actual argument lists, type parameter lists, and structure constructors, items may be identified by a preceding *keyword*= rather than their position within the list. An **argument keyword** is the name of a dummy argument in the interface for the procedure being referenced, a **type parameter keyword** is the name of a type parameter in the type being specified, and a **component keyword** is the name of a component in a structure constructor.

R215 *keyword* **is** *name*

NOTE 2.6

Use of keywords rather than position to identify items in a list can make such lists more readable and allows them to be reordered. This facilitates specification of a list in cases where optional items are omitted.

2.5.3 Association

Association is name association (16.4.1), pointer association (16.4.2), storage association (16.4.3), or inheritance association (16.4.4). Name association is argument association, host association, use association, linkage association, or construct association.

Storage association causes different entities to use the same storage. Any association permits an entity to be identified by different names in the same scoping unit or by the same name or different names in different scoping units.

2.5.4 Declaration

The term **declaration** refers to the specification of attributes for various program entities. Often this involves specifying the type of a named data object or specifying the shape of a named array object.

2.5.5 Definition

The term **definition** is used in two ways.

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

(1) It refers to the specification of derived types and procedures.

(2) When an object is given a valid value during program execution, it is said to become **defined**. This is often accomplished by execution of an assignment or input statement. When a variable does not have a predictable value, it is said to be **undefined**. Similarly, when a pointer is associated with a target or nullified, its pointer association status is said to become **defined**. When the association status of a pointer is not predictable, its pointer association status is said to be **undefined**.

Section 16 describes the ways in which variables may become defined and undefined.

2.5.6 Reference

A **data object reference** is the appearance of the data object designator in a context requiring its value at that point during execution.

A **procedure reference** is the appearance of the procedure designator, operator symbol, or assignment symbol in a context requiring execution of the procedure at that point. An occurrence of user-defined derived-type input/output (10.6.5) or derived-type finalization (4.5.5.1) is also a procedure reference.

The appearance of a data object designator or procedure designator in an actual argument list does not constitute a reference to that data object or procedure unless such a reference is necessary to complete the specification of the actual argument.

A **module reference** is the appearance of a module name in a USE statement (11.2.1).

2.5.7 Intrinsic

The qualifier **intrinsic** has two meanings.

(1) The qualifier signifies that the term to which it is applied is defined in this standard. Intrinsic applies to types, procedures, modules, assignment statements, and operators. All intrinsic types, procedures, assignments, and operators may be used in any scoping unit without further definition or specification. Intrinsic modules may be accessed by use association. Intrinsic procedures and modules defined in this standard are called standard intrinsic procedures and standard intrinsic modules, respectively.

(2) The qualifier applies to procedures or modules that are provided by a processor but are not defined in this standard (13, 14, 15.1). Such procedures and modules are called nonstandard intrinsic procedures and nonstandard intrinsic modules, respectively.

2.5.8 Operator

An **operator** specifies a computation involving one (unary operator) or two (binary operator) data values (**operands**). This standard specifies a number of intrinsic operators (e.g., the arithmetic operators +, –, *, /, and ** with numeric operands and the logical operators .AND., .OR., etc. with logical operands). Additional operators may be defined within a program (7.1.3).

2.5.9 Sequence

A **sequence** is a set ordered by a one-to-one correspondence with the numbers 1, 2, through n. The number of elements in the sequence is n. A sequence may be empty, in which case it contains no elements.

The elements of a nonempty sequence are referred to as the first element, second element, etc. The nth element, where n is the number of elements in the sequence, is called the last element. An empty sequence has no first or last element.

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

2.5.10 Companion processors

A processor has one or more companion processors. A **companion processor** is a processor-dependent mechanism by which global data and procedures may be referenced or defined. A companion processor may be a mechanism that references and defines such entities by a means other than Fortran (12.5.3), it may be the Fortran processor itself, or it may be another Fortran processor. If there is more than one companion processor, the means by which the Fortran processor selects among them are processor dependent.

If a procedure is defined by means of a companion processor that is not the Fortran processor itself, this standard refers to the C function that defines the procedure, although the procedure need not be defined by means of the C programming language.

NOTE 2.7

A companion processor might or might not be a mechanism that conforms to the requirements of the C International Standard.

For example, a processor may allow a procedure defined by some language other than Fortran or C to be invoked if it can be described by a C prototype as defined in 6.5.5.3 of the C International Standard.

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

(Blank page)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Section 3: Characters, lexical tokens, and source form

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

3.1 Processor character set

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

(1) **Control characters**
(2) **Graphic characters**
  (a) Letters (3.1.1)
  (b) Digits (3.1.2)
  (c) Underscore (3.1.3)
  (d) Special characters (3.1.4)
  (e) Other characters (3.1.5)

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

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

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

The default character type shall support a character set that includes the Fortran character set. By supplying nondefault character types, the processor may support additional character sets. The characters available in the ASCII and ISO 10646 character sets are specified by ISO/IEC 646:1991 (International Reference Version) and ISO/IEC 10646-1:2000 UCS-4, respectively; the characters available in other nondefault character types are not specified by this standard, except that one character in each nondefault character type shall be designated as a blank character to be used as a padding character.

3.1.1 Letters

The twenty-six **letters** are:

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

The set of letters defines the syntactic class *letter*. The processor character set shall include lower-case and upper-case letters. A lower-case letter is equivalent to the corresponding upper-case letter in program units except in a character context (3.3).

NOTE 3.1

The following statements are equivalent:

```
CALL BIG_COMPLEX_OPERATION (NDATE)
call big_complex_operation (ndate)
```


Documents
Document Sharing Hub
Full document details: BS ISO/IEC 1539-1:2004

NOTE 3.1 (cont.)

```
Call Big_Complex_Operation (NDate)
```

3.1.2 Digits

The ten **digits** are:

0 1 2 3 4 5 6 7 8 9

The ten digits define the syntactic class *digit*.

3.1.3 Underscore

R303 *underscore* **is** _

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

3.1.4 Special characters

The **special characters** are shown in Table 3.1.

Table 3.1: **Special characters**

Character	Name of character	Character	Name of character
	Blank	;	Semicolon
=	Equals	!	Exclamation point
+	Plus	"	Quotation mark or quote
-	Minus	%	Percent
*	Asterisk	&	Ampersand
/	Slash	~	Tilde
\	Backslash	<	Less than
(	Left parenthesis	>	Greater than
)	Right parenthesis	?	Question mark
[	Left square bracket	’	Apostrophe
]	Right square bracket	`	Grave accent
{	Left curly bracket	^	Circumflex accent
}	Right curly bracket	|	Vertical line
,	Comma	$	Currency symbol
.	Decimal point or period	#	Number sign
:	Colon	@	Commercial at

The special characters define the syntactic class *special-character*. Some of the special characters are used for operator symbols, bracketing, and various forms of separating and delimiting other lexical tokens.

3.1.5 Other characters

Additional characters may be representable in the processor, but may appear only in comments (3.3.1.1, 3.3.2.1), character constants (4.4.4), input/output records (9.1.1), and character string edit descriptors (10.2.1).

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

3.2 Low-level syntax

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

3.2.1 Names

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

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

C301 (R304) The maximum length of a *name* is 63 characters.

NOTE 3.2

Examples of names:

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

NOTE 3.3

The word "name" always denotes this particular syntactic form. The word "identifier" is used where entities may be identified by other syntactic forms or by values; its particular meaning depends on the context in which it is used.

3.2.2 Constants

R305 *constant* **is** *literal-constant*
or *named-constant*

R306 *literal-constant* **is** *int-literal-constant*
or *real-literal-constant*
or *complex-literal-constant*
or *logical-literal-constant*
or *char-literal-constant*
or *boz-literal-constant*

R307 *named-constant* **is** *name*

R308 *int-constant* **is** *constant*

C302 (R308) *int-constant* shall be of type integer.

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

C303 (R309) *char-constant* shall be of type character.

3.2.3 Operators

R310 *intrinsic-operator* **is** *power-op*
or *mult-op*
or *add-op*
or *concat-op*

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

		or	*rel-op*
		or	*not-op*
		or	*and-op*
		or	*or-op*
		or	*equiv-op*
R707	*power-op*	**is**	**
R708	*mult-op*	**is**	*
		or	/
R709	*add-op*	**is**	+
		or	–
R711	*concat-op*	**is**	//
R713	*rel-op*	**is**	.EQ.
		or	.NE.
		or	.LT.
		or	.LE.
		or	.GT.
		or	.GE.
		or	==
		or	/=
		or	<
		or	<=
		or	>
		or	>=
R718	*not-op*	**is**	.NOT.
R719	*and-op*	**is**	.AND.
R720	*or-op*	**is**	.OR.
R721	*equiv-op*	**is**	.EQV.
		or	.NEQV.
R311	*defined-operator*	**is**	*defined-unary-op*
		or	*defined-binary-op*
		or	*extended-intrinsic-op*
R703	*defined-unary-op*	**is**	. *letter* [*letter*] ... .
R723	*defined-binary-op*	**is**	. *letter* [*letter*] ... .
R312	*extended-intrinsic-op*	**is**	*intrinsic-operator*

3.2.4 Statement labels

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

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

C304 (R313) At least one digit in a *label* shall be nonzero.

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

NOTE 3.4

For example:

```
99999
10
 010
```


Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

NOTE 3.4 (cont.)

are all statement labels. The last two are equivalent.

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

Any statement may have a statement label, but the labels are used only in the following ways:

(1) The label on a branch target statement (8.2) is used to identify that statement as the possible destination of a branch.

(2) The label on a FORMAT statement (10.1.1) is used to identify that statement as the format specification for a data transfer statement (9.5).

(3) In some forms of the DO construct (8.1.6), the range of the DO construct is identified by the label on the last statement in that range.

3.2.5 Delimiters

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

```
( ... )
/ ... /
[ ... ]
(/ ... /)
```

3.3 Source form

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

A **character context** means characters within a character literal constant (4.4.4) or within a character string edit descriptor (10.2.1).

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

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

3.3.1 Free source form

In **free source form** there are no restrictions on where a statement (or portion of a statement) may appear within a line. A line may contain zero characters. If a line consists entirely of characters of default kind (4.4.4), it may contain at most 132 characters. If a line contains any character that is not of default kind, the maximum number of characters allowed on the line is processor dependent.

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

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

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

NOTE 3.5

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

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

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

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

3.3.1.1 Free form commentary

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

NOTE 3.6

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

3.3.1.2 Free form statement continuation

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

If a noncharacter context is to be continued, an "&" shall be the last nonblank character on the line, or the last nonblank character before an "!". There shall be a later line that is not a comment; the statement is continued on the next such line. If the first nonblank character on that line is an "&", the statement continues at the next character position following that "&"; otherwise, it continues with the first character position of that line.

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

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

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

3.3.1.3 Free form statement termination

If a statement is not continued, a comment or the end of the line terminates the statement.

A statement may alternatively be terminated by a ";" character that appears other than in a character context or in a comment. The ";" is not part of the statement. After a ";" terminator, another statement may appear on the same line, or begin on that line and be continued. A ";" shall not appear as the first nonblank character on a line. A sequence consisting only of zero or more blanks and one or more ";" terminators, in any order, is equivalent to a single ";" terminator.

3.3.1.4 Free form statements

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

NOTE 3.7

No Fortran statement begins with a digit.

A statement shall not have more than 255 continuation lines.

3.3.2 Fixed source form

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

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

3.3.2.1 Fixed form commentary

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

NOTE 3.8

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

3.3.2.2 Fixed form statement continuation

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

NOTE 3.9

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

AVA Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

Comment lines cannot be continued. Comment lines may occur within a continued statement.

3.3.2.3 Fixed form statement termination

If a statement is not continued, a comment or the end of the line terminates the statement.

A statement may alternatively be terminated by a ";" character that appears other than in a character context, in a comment, or in character position 6. The ";" is not part of the statement. After a ";" terminator, another statement may begin on the same line, or begin on that line and be continued. A ";" shall not appear as the first nonblank character on a line, except in character position 6. A sequence consisting only of zero or more blanks and one or more ";" terminators, in any order, is equivalent to a single ";" terminator.

3.3.2.4 Fixed form statements

A label, if present, shall occur in character positions 1 through 5 of the first line of a statement; otherwise, positions 1 through 5 shall be blank. Blanks may appear anywhere within a label. A statement following a ";" on the same line shall not be labeled. Character positions 1 through 5 of any continuation lines shall be blank. A statement shall not have more than 255 continuation lines. The program unit END statement shall not be continued. A statement whose initial line appears to be a program unit END statement shall not be continued.

3.4 Including source text

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

INCLUDE *char-literal-constant*

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

An INCLUDE line is not a Fortran statement.

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

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

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

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

NOTE 3.10

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

(1) Confine statement labels to character positions 1 to 5 and statements to character positions 7 to 72;
(2) Treat blanks as being significant;

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

NOTE 3.10 (cont.)

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

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

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004

(Blank page)

Documents
Document Sharing Hub

Full document details: BS ISO/IEC 1539-1:2004