Annex A

(informative)

Glossary of technical terms

The following is a list of the principal technical terms used in the standard and their definitions. A reference in parentheses immediately after a term is to the section where the term is defined or explained. The wording of a definition here is not necessarily the same as in the standard. Where the definition uses a term that is itself defined in this glossary, the first occurrence of the term in that definition is printed in italics.

action statement (2.1) : A single *statement* specifying or controlling a computational action (R216).

actual argument (12.4.1) : An *expression*, a *variable*, a *procedure*, or an alternate return specifier that is specified in a procedure *reference*.

allocatable array (5.1.2.4.3) : A named *array* having the ALLOCATABLE *attribute*. Only when it has space allocated for it does it have a *shape* and may it be *referenced* or *defined*.

argument (12) : An *actual argument* or a *dummy argument*.

argument association (14.6.1.1) : The relationship between an *actual argument* and a *dummy argument* during the execution of a *procedure reference*.

argument keyword (2.5.2) : A *dummy argument name*. It may be used in a *procedure reference* followed by the equals symbol (R1212) provided the procedure has an *explicit interface*.

array (2.4.5) : A set of scalar *data*, all of the same *type* and *type parameters*, whose individual elements are arranged in a rectangular pattern. It may be a *named* array, an *array section*, a *structure component*, a *function* value, or an *expression*. Its *rank* is at least one. Note that in Fortran 77, arrays were always named and never constants.

array element (2.4.5, 6.2.2) : One of the *scalar data* that make up an *array* that is either *named* or is a *structure component*.

array pointer (5.1.2.4.3) : A *pointer* to an *array*.

array section (2.4.5, 6.2.2.3) : A *subobject* that is an *array* and is not a *structure component*.

array-valued : Having the property of being an *array*.

assignment statement (7.5.1.1) : A *statement* of the form "*variable* = *expression*".

association (14.6) : Name association, pointer association, or storage association.

assumed-shape array (5.1.2.4.2) : A nonpointer dummy array that takes its shape from the associated actual argument.

assumed-size array (5.1.2.4.4) : A *dummy array* whose *size* is assumed from the associated *actual argument*. Its last upper bound is specified by an asterisk.

attribute (5) : A property of a *data object* that may be specified in a *type declaration statement* (R501).

automatic data object (5.1) : A *data object* that is a *local entity* of a *subprogram*, that is not a *dummy argument*, and that has a nonconstant character length or array bound.

belong (8.1.4.4.3, 8.1.4.4.4) : If an EXIT or a CYCLE *statement* contains a *construct name*, the statement **belongs** to the DO construct using that name. Otherwise, it **belongs** to the innermost DO construct in which it appears.

block (8.1) : A sequence of *executable constructs* embedded in another executable construct, bounded by *statements* that are particular to the construct, and treated as an integral unit.

block data program unit (11.4) : A *program unit* that provides initial values for *data objects* in *named common blocks*.

bounds (5.1.2.4.1) : For a *named array*, the limits within which the values of the *subscripts* of itsarray *elements* shall lie.

character (3.1) : A letter, digit, or other symbol.

characteristics (12.2) :

(1) Of a *procedure*, its classification as a *function* or *subroutine*, the characteristics of its *dummy arguments*, and the characteristics of its *function result* if it is a function.

(2) Of a *dummy argument*, whether it is a *data object*, is a *procedure*, or has the OPTIONAL *attribute*.

(3) Of a *data object*, its *type*, *type parameters*, *shape*, the exact dependence of an array bound or the character length on other entities, *intent*, whether it is optional, whether it is a *pointer* or a *target*, and whether the shape, *size*, or *character length* is assumed.

(4) Of a *dummy procedure*, whether the interface is explicit, the characteristics of the procedure if the interface is explicit, and whether it is optional.

(5) Of a *function result*, its type, type parameters, whether it is a pointer, rank if it is a pointer, shape if it is not a pointer, the exact dependence of an array bound or the character length on other entities, and whether the character length is assumed.

character length parameter (2.4.1.1) : The *type parameter* that specifies the number of characters for an *entity* of type character.

character string (4.3.2.1) : A sequence of *characters* numbered from left to right 1, 2, 3, ...

character storage unit (14.6.3.1) : The unit of storage for holding a scalar that is not a *pointer* and is of *type* default character and character length one.

collating sequence (4.3.2.1.1) : An ordering of all the different *characters* of a particular *kind type parameter*.

common block (5.5.2) : A block of physical storage that may be accessed by any of the *scoping units* in a *program*.

component (4.4) : A constituent of a *derived type*.

conformable (2.4.5) : Two *arrays* are said to be **conformable** if they have the same *shape*. A *scalar* is conformable with any array.

conformance (1.5) : A *program* conforms to the standard if it uses only those forms and relationships described therein and if the program has an interpretation according to the standard. A *program unit* conforms to the standard if it can be included in a program in a manner that allows the program to be standard conforming. A *processor* conforms to the standard if it executes standard-conforming programs in a manner that fulfills the interpretations prescribed in the standard and contains the capability of detection and reporting as listed in 1.5.

connected (9.3.2) :

(1) For an *external unit*, the property of referring to an *external file*.

(2) For an external file, the property of having an external unit that refers to it.

constant (2.4.3.1.2) : A *data object* whose value shall not change during execution of a *program*. It may be a *named constant* or a *literal constant*.

constant expression (7.1.6.1) : An *expression* satisfying rules that ensure that its value does not vary during *program* execution.

construct (7.5.3, 7.5.4, 8.1) : A sequence of *statements* starting with a SELECT CASE, DO, IF, FORALL, or WHERE statement and ending with the corresponding terminal statement.

construct entity (14) : An *entity* defined by a *lexical token* whose *scope* is a *construct*.

control mask (7.5.3) : In a WHERE *statement* or *construct*, an *array* of *type* logical whose value determines which elements of an array, in a *where-assignment-stmt*, will be defined.

data : Plural of *datum*.

data entity (2.4.3) : 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 *data type* (either *intrinsic* or *derived*) and has, or may have, a data value (the exception is an *undefined variable*). Every data entity has a *rank* and is thus either a *scalar* or an *array.*

data object (2.4.3.1) : A *data entity* that is a *constant*, a *variable*, or a *subobject* of a constant.

data type (2.4.1) : A *named* category of *data* that is characterized by a set of values, together with a way to denote these values and a collection of *operations* that interpret and manipulate the values. For an *intrinsic* type, the set of data values depends on the values of the *type parameters*.

datum : A single quantity that may have any of the set of values specified for its *data type*.

default initialization (4.4) : If initialization is specified in a type definition, an object of the type will be automatically initialized. Nonpointer components may be initialized with values by default; pointer components may be initially disassociated by default. Default initialization is not provided for objects of intrinsic type.

definable (2.5.4) : A *variable* is **definable** if its value may be changed by the appearance of its *name* or *designator* on the left of an *assignment statement*. An *allocatable array* that has not been allocated is an example of a *data object* that is not definable. An example of a *subobject* that is not definable is C (I) when C is an *array* that is a *constant* and I is an integer variable.

defined (2.5.4) : For a *data object*, the property of having or being given a valid value.

defined assignment statement (7.5.1.3) : An *assignment statement* that is not an *intrinsic* assignment statement and is defined by a *subroutine* and an *interface block* that specifies ASSIGNMENT (=).

defined operation (7.1.3) : An *operation* that is not an *intrinsic* operation and is defined by a *function* that is associated with a *generic identifier.*

deleted feature (1.7) : A feature in a previous Fortran standard that is considered to have been redundant and largely unused. See section B.1 for a list of features which were in a previous Fortran standard, but are not in this standard. A feature designated as an *obsolescent feature* in the standard may become a deleted feature in the next revision.

derived type (2.4.1.2, 4.4) : A *type* whose *data* have *components*, each of which is either of *intrinsic* type or of another derived type.

designator : See *subobject designator.*

direct component (4.4) : The direct components of a *derived type* are

(1) The *components* of that *type* and

(2) For any nonpointer component that is of derived type, the direct components of that derived type.

disassociated (2.4.6) : A *pointer* is **disassociated** following execution of a DEALLOCATE or NULLIFY *statement*, or following *pointer association* with a disassociated pointer.

dummy argument (12.5.2.2, 12.5.2.3, 12.5.2.5, 12.5.4) : An entity whose *name* appears in the parenthesized list following the *procedure* name in a FUNCTION *statement*, a SUBROUTINE *statement*, an ENTRY statement, or a *statement function* statement.

dummy array : A *dummy argument* that is an *array*.

dummy pointer : A *dummy argument* that is a *pointer*.

dummy procedure (12.1.2.3) : A *dummy argument* that is specified or *referenced* as a *procedure*.

elemental (2.4.5, 7.5.1.3, 12.7) : An adjective applied to an *operation*, *procedure*, or *assignment statement* that is applied independently to elements of an *array* or corresponding elements of a set of *conformable* arrays and *scalars*.

entity : The term used for any of the following: a *program unit*, a *procedure*, an *operator*, an *interface block*, a *common block*, an *external unit*, a *statement function*, a *type*, a *data entity*, a *statement label*, a *construct*, or a namelist group.

executable construct (2.1) : A CASE, DO, FORALL, IF, or WHERE *construct* or an *action statement* (R216).

executable statement (2.3.1) : An instruction to perform or control one or more computational actions.

explicit initialization (5.1) : Explicit initialization may be specified for objects of intrinsic or derived type in type declaration statements or DATA statements. An object of a derived type that specifies *default initialization* may not appear in a DATA statement.

explicit interface (12.3.1) : For a *procedure referenced* in a *scoping unit*, the property of being an *internal procedure*, a *module procedure*, an *intrinsic procedure*, an *external procedure* that has an *interface block*, a recursive procedure reference in its own scoping unit, or a *dummy procedure* that has an interface block.

explicit-shape array (5.1.2.4.1) : A *named array* that is declared with explicit *bounds*.

expression (2.4.3.2, 7.1) : A sequence of *operands*, *operators*, and parentheses (R723). It may be a *variable*, a *constant*, a *function reference*, or may represent a computation.

extent (2.4.5) : The size of one dimension of an *array*.

external file (9.2.1) : A sequence of *records* that exists in a medium external to the *program*.

external procedure (2.2.3.1) : A *procedure* that is defined by an *external subpr* or by a means other than Fortran.

external subprogram (2.2) : A *subprogram* that is not in a *main program*, *module*, or another subprogram. Note that a module is not called a subprogram. Note that in FORTRAN 77, a *block data program unit* is called a subprogram.

external unit (9.3) : A mechanism that is used to refer to an *external file*. It is identified by a nonnegative integer.

file (9.2) : An *internal file* or an *external file*.

function (2.2.3) : A *procedure* that is invoked in an *expression* and computes a value which is then used in evaluating the expression.

function result (12.5.2.2) : The *data object* that returns the value of a *function*.

function subprogram (12.5.2.2) : A sequence of *statements* beginning with a FUNCTION statement that is not in an *interface block* and ending with the corresponding END statement.

generic identifier (12.3.2.1) : A *lexical token* that appears in an INTERFACE *statement* and is associated with all the *procedures* in the *interface block*.

global entity (14.1.1) : An *entity* identified by a *lexical token* whose *scope* is a *program*. It may be a *program unit*, a *common block*, or an *external procedure*.

host (2.2.3.2, 2.2.3.3) : A *main program* or *subprogram* that contains an *internal subprogram* is called the **host** of the internal subprogram. A *module* that contains a *module subprogram* is called the **host** of the module subprogram.

host association (14.6.1.3) : The process by which an *internal subprogram*, *module subprogram*, or *derived type* definition accesses *entities* of its *host*.

host scoping unit (2.2) : A *scoping unit* that immediately surrounds another scoping unit.

implicit interface (12.3.1) : A *procedure referenced* in a *scoping unit* other than its own is said to have an implicit interface if the procedure is an *external procedure* that does not have an *interface block*, a *dummy procedure* that does not have an interface block, or a *statement function*.

inquiry function (13.1) : An *intrinsic function* whose result depends on properties of the principal *argument* other than the value of the argument.

intent (12.5.2.1) : An attribute of a *dummy argument* that is neither a *procedure* nor a *pointer*, which indicates whether it is used to transfer data into the procedure, out of the procedure, or both.

instance of a subprogram (12.5.2.4) : The copy of a *subprogram* that is created when a *procedure* defined by the subprogram is *invoked*.

interface block (12.3.2.1) : A sequence of *statements* from an INTERFACE statement to the corresponding END INTERFACE statement.

interface body (12.3.2.1) : A sequence of *statements* in an *interface block* from a FUNCTION or SUBROUTINE statement to the corresponding END statement.

interface of a procedure (12.3) : See *procedure interface*.

internal file (9.2.2) : A character *variable* that is used to transfer and convert *data* from internal storage to internal storage.

internal procedure (2.2.3.3) : A *procedure* that is defined by an *internal subprogram*.

internal subprogram (2.2) : A *subprogram* in a *main program* or another subprogram.

intrinsic (2.5.7) : An adjective applied to *data types*, *operations*, *assignment statements*, and *procedures* that are defined in the standard and may be used in any *scoping unit* without further definition or specification.

invoke (2.2.3) :

(1) To call a *subroutine* by a CALL *statement* or by a *defined assignment statement.*

(2) To call a *function* by a *reference* to it by *name* or *operator* during the evaluation of an *expression*.

keyword (2.5.2) : *Statement keyword* or *argument keyword*.

kind type parameter (2.4.1.1, 4.3.1.1, 4.3.1.2, 4.3.1.3, 4.3.2.1, 4.3.2.2) : A parameter whose values label the available kinds of an *intrinsic type*.

label : See *statement label*.

length of a character string (4.3.2.1) : The number of *characters* in the *character string*.

lexical token (3.2) : A sequence of one or more characters with a specified interpretation.

line (3.3) : A sequence of 0 to 132 *characters*, which may contain Fortran *statements*, a comment, or an INCLUDE line.

literal constant (2.4.3.1.2, 4.3) : A *constant* without a *name*. Note that in FORTRAN 77, this was called simply a constant.

local entity (14.1.2) : An *entity* identified by a *lexical token* whose *scope* is a *scoping unit*.

main program (11.1) : A *program unit* that is not a *module*, *external subprogram*, or *block data program unit*.

many-one array section (6.2.2.3.2) : An *array section* with a *vector subscript* having two or more elements with the same value.

module (2.2.4, 11.3) : A *program unit* that contains or accesses definitions to be accessed by other program units.

module procedure (2.2.3.2) : A *procedure* that is defined by a *module subprogram*.

module subprogram (2.2) : A *subprogram* that is in a *module* but is not an *internal subprogram*.

name (3.2.1) : A *lexical token* consisting of a letter followed by up to 30 alphanumeric characters (letters, digits, and underscores). Note that in FORTRAN 77, this was called a symbolic name.

name association (14.6.1) : Argument association, use association, or host association.

named : Having a *name*. That is, in a phrase such as "named variable," the word "named" signifies that the variable name is not qualified by a subscript list, substring specification, and so on. For example, if X is an array variable, the reference "X" is a named variable while the reference "X(1)" is a subobject designator.

named constant (2.4.3.1.2) : A *constant* that has a *name*. Note that in FORTRAN 77, this was called a symbolic constant.

nonexecutable statement (2.3.1) : A statement used to configure the program environment in which computational actions take place.

numeric storage unit (14.6.3.1) : The unit of storage for holding a *scalar* that is not a *pointer* and is of *type* default real, default integer, or default logical.

numeric type (4.3.1) : Integer, real or complex *type*.

object (2.4.3.1) : *Data object*.

obsolescent feature (1.7) : A feature that is considered to have been redundant but that is still in frequent use.

operand (2.5.8) : An *expression* that precedes or succeeds an *operator*.

operation (7.1.2) : A computation involving one or two *operands*.

operator (2.5.8) : A *lexical token* that specifies an *operation*.

override (4.4.1) : When *explicit initialization* or *default initialization* overrides default initialization, it is as if only the overriding initialization were specified.

pointer (2.4.6) : A *variable* that has the POINTER attribute. A pointer shall not be *referenced* or *defined* unless it is *pointer associated* with a *target*. If it is an *array*, it does not have a *shape* unless it is *pointer associated*, although it does have a rank.

pointer assignment (7.5.2) : The *pointer association* of a *pointer* with a *target* by the execution of a *pointer assignment statement* or the execution of an *assignment statement* for a *data object* of *derived type* having the pointer as a *subobject*.

pointer assignment statement (7.5.2) : A *statement* of the form "*pointer-object => target*".

pointer associated (6.3, 7.5.2) : The relationship between a *pointer* and a *target* following a *pointer assignment* or a valid execution of an ALLOCATE *statement*.

pointer association (14.6.2) : The process by which a *pointer* becomes *pointer associated* with a *target*.

preconnected (9.3.3) : A property describing a unit that is connected to an *external file* at the beginning of execution of a *program*. Such a unit may be specified in input/output statements without an OPEN statement being executed for that unit.

present (12.4.1.5) : A *dummy argument* is **present** in an *instance of a subprogram* if it is *associated* with an *gument* and the actual argument is a dummy argument that is present in the invoking *subprogram* or is not a dummy argument of the invoking subprogram.

procedure (2.2.3, 12.1) : A computation that may be *invoked* during *program* execution. It may be a *function* or a *subroutine*. It may be an *intrinsic* procedure, an *external procedure*, a *module procedure*, an *internal procedure*, a *dummy procedure*, or a *statement function*. A *subprogram* may define more than one procedure if it contains ENTRY *statements*.

procedure interface (12.3) : The *characteristics* of a *procedure*, the *name* of the procedure, the name of each *dummy argument*, and the *generic identifiers* (if any) by which it may be *referenced*.

processor (1.2) : The combination of a computing system and the mechanism by which *programs* are transformed for use on that computing system.

processor dependent (1.5) : The designation given to a facility that is not completely specified by this standard. Such a facility shall be provided by a *processor*, with methods or semantics determined by the processor.

program (2.2.1) : A set of *program units* that includes exactly one *main program*.

program unit (2.2) : The fundamental component of a *program*. A sequence of *statements*, comments, and INCLUDE lines. It may be a *main program*, a *module*, an *external subprogram*, or a *block data program unit*.

rank (2.4.4, 2.4.5) : The number of dimensions of an *array*. Zero for a *scalar*.

record (9.1) : A sequence of values or characters that is treated as a whole within a *file*.

reference (2.5.5) : The appearance of a *data object name* or *subobject designator* in a context requiring the value at that point during execution, the appearance of a *procedure* name, its *operator* symbol, or a *defined assignment statement* in a context requiring execution of the procedure at that point, or the appearance of a *module* name in a USE statement. Neither the act of defining a *variable* nor the appearance of the name of a procedure as an *actual argument* is regarded as a reference.

result variable (2.2.3, 12.5.2.2) : The *variable* that returns the value of a *function*.

scalar (2.4.4) :

(1) A single *datum* that is not an *array*.

(2) Not having the property of being an *array*.

scope (14) : That part of a *program* within which a *lexical token* has a single interpretation. It may be a program, a *scoping unit*, a *construct*, a single *statement*, or a part of a statement.

scoping unit (2.2) : One of the following:

(1) A *derived type* definition,

(2) An *interface body*, excluding any derived-type definitions and interface bodies in it, or

(3) A *program unit* or *subprogram*, excluding derived-type definitions, interface bodies, and subprograms in it.

section subscript (6.2.2) : A *subscript*, *vector subscript*, or *subscript triplet* in an *array section selector*.

selector : A syntactic mechanism for designating

(1) Part of a *data object*. It may designate a *substring*, an *array element*, an *array section*, or a *structure component*.

(2) The set of values for which a CASE *block* is executed.

shape (2.4.5) : For an *array*, the *rank* and *extents*. The shape may be represented by the rank-one array whose elements are the extents in each dimension.

size (2.4.5) : For an *array*, the total number of elements.

standard-conforming program (1.5) : A *program* that uses only those forms and relationships described in this standard, and which has an interpretation according to this standard.

statement (3.3) : A sequence of *lexical tokens*. It usually consists of a single line, but a statement may be continued from one line to another and the semicolon symbol may be used to separate statements within a line.

statement entity (14) : An *entity* identified by a *lexical token* whose *scope* is a single *statement* or part of a statement.

statement function (12.5.4) : A *procedure* specified by a single *statement* that is similar in form to an *assignment statement*.

statement keyword (2.5.2) : A word that is part of the syntax of a *statement* and that may be used to identify the statement.

statement label (3.2.4) : A *lexical token* consisting of up to five digits that precedes a *statement* and may be used to refer to the statement.

storage association (14.6.3) : The relationship between two *storage sequences* if a storage unit of one is the same as a storage unit of the other.

storage sequence (14.6.3.1) : A sequence of contiguous *storage units*.

storage unit (14.6.3.1) : A *character storage unit*, a *numeric storage unit*, or an *unspecified storage unit*.

stride (6.2.2.3.1) : The increment specified in a *subscript triplet*.

structure (2.4.1.2) : A *scalar data object* of *derived type*.

structure component (6.1.2) : A part of an *object* of *derived type* that may be referenced by a *subobject designator*.

subobject (2.4.3.1) : A portion of a *named data object* that may be *referenced* or *defined* independently of other portions. It may be an *array element*, an *array section*, a *structure component*, or a *substring*.

subobject designator (2.5.1) : A *name*, followed by one or more of the following: *component selectors*, *array section* selectors, *array element* selectors, and *substring* selectors.

subprogram (2.2) : A *function subprogram* or a *subroutine subprogram*. Note that in FORTRAN 77, a *block data program unit* was called a subprogram.

subroutine (2.2.3) : A *procedure* that is *invoked* by a CALL *statement* or by a *defined assignment statement*.

subroutine subprogram (12.5.2.3) : A sequence of *statements* beginning with a SUBROUTINE statement that is not in an *interface block* and ending with the corresponding END statement.

subscript (6.2.2) : One of the list of *scalar* integer *expressions* in an *array element selector*. Note that in FORTRAN 77, the whole list was called the subscript.

subscript triplet (6.2.2) : An item in the list of an *array section selector* that contains a colon and specifies a regular sequence of integer values.

substring (6.1.1) : A contiguous portion of a *scalar character string*. Note that an *array section* can include a *substring selector*; the result is called an array section and not a substring.

target (5.1.2.8) : A *named data object* specified in a TARGET statement or in a *type declaration statement* containing the TARGET *attribute*, a data object created by an ALLOCATE statement for a pointer, or a *subobject* of such an object.

transformational function (13.1) : An *intrinsic function* that is neither an *elemental* function nor an *inquiry function*. It usually has *array arguments* and an array result whose elements have values that depend on the values of many of the elements of the arguments.

type (4) *:* Data type.

type declaration statement (5) : An INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER, LOGICAL, or TYPE (*type-name*) *statement*.

type parameter (2.4.1.1) : A parameter of an *intrinsic data type*. KIND and LEN are the type parameters.

type parameter values (4.3) : The values of the *type parameters* of a *data entity* of an *intrinsic data type*.

ultimate component (4.4) : For a *derived type* or a *structure*, a *component* that is of *intrinsic type* or has the POINTER attribute, or an *ultimate component* of a component that is a derived type and does not have the POINTER attribute.

undefined (2.5.4) : For a *data object*, the property of not having a determinate value.

unspecified storage unit (14.6.3.1) : A unit of storage for holding a *pointer* or a *scalar* that is not a pointer and is of *type* other than default integer, default character, default real, double precision real, default logical, or default complex.

use association (14.6.1.2) : The association of *names* in different *scoping units* specified by a USE *statement*.

variable (2.4.3.1.1) : A *data object* whose value can be *defined* and redefined during the execution of a *program*. It may be a *named* data object, an *array element*, an *array section*, a *structure component*, or a *substring*. Note that in FORTRAN 77, a variable was always *scalar* and named.

vector subscript (6.2.2.3.2) : A *section subscript* that is an integer *expression* of *rank* one.

whole array (6.2.1) : A *named array*.