Reference number of working document: **J3/12-181**

Date: 2012-10-05

Reference number of document: **ISO/IEC TS 99999:2012(E)**

Committee identification: ISO/IEC JTC1/SC22

Secretariat: ANSI

Information technology — Programming languages — Fortran — Abstract subprograms

Technologies de l'information — Langages de programmation — Fortran — Sous-programmes abstraits

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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

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

ISO/IEC TS 99999:2012(E) was prepared by Joint Technical Committee ISO/IEC/JTC1, *Information technology*, Subcommittee SC22, *Programming languages, their environments and system software interfaces*.

This technical specification specifies an extension to the computational facilities of the programming language Fortran. Fortran is specified by the International Standard ISO/IEC 1539-1:2010(E).

It is the intention of ISO/IEC JTC1/SC22/WG5 that the semantics and syntax specified by this technical specification be included in the next revision of the Fortran International Standard without change unless experience in the implementation and use of this feature identifies errors that need to be corrected, or changes are needed to achieve proper integration, in which case every reasonable effort will be made to

minimize the impact of such changes on existing implementations.

0 Introduction

0.1 History

Since Fortran 2003, derived types can be parameterized by kind type parameters, and can have type-bound procedures with generic bindings. Where a type-bound procedure is invoked, if its binding does not have the NOPASS attribute, the object used to invoke it is associated as an actual argument. If one has declared an object using kind type parameters such that no specific type-bound procedure has appropriate kind type parameters for its arguments, a violation of a constraint exists.

Even if one limits attention to kind type parameters for intrinsic types defined by ISO/IEC 1539-1:2010(E), it is tedious and sometimes difficult to ensure that all necessary type-bound procedures exist to correspond to every possible declaration of objects of the type. It is not possible, in general, to anticipate all kind type parameters of intrinsic types that are offered as processor extensions.

0.2 What this technical specification proposes

This technical specification proposes to extend the syntax of definition of subprograms to allow to define an abstract subprogram. An abstract subprogram is a definition of a family of programs. An abstract subprogram cannot be invoked. Instead, one can instantiate a member of that family by specifying parameters by constant integer expressions. Once that member has been instantiated, that instantiation can be invoked.

Information technology – Programming Languages – Fortran

Technical Specification: Abstract subprograms

1 General

1.1 Scope

This technical specification specifies an extension to the programming language Fortran. The Fortran language is specified by International Standard ISO/IEC 1539-1:2010(E). The extension consists of an extension to the syntax to allow to define an abstract subprogram, and to create an instantiation of it that is parameterized by a set of constant integer expressions. An instantiations of an abstract procedure behaves in all respects but one in exactly the same ways as a subprogram defined by International Standard ISO/IEC 1539-1:2010(E). The single exception is that an instantiation of an abstract subprogram does not access the scoping unit containing its instantiation by host association; rather, it accesses the scoping unit containing the definition of the abstract subprogram of which it is an instantiation by host association.

Clause 2 of this technical specification contains a general and informal but precise description of the extended functionalities. Clause 3 contains several illustrative examples. Clause 4 contains detailed instructions for editorial changes to ISO/IEC 1539-1:2010(E).

1.2 Normative References

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

ISO/IEC 1539-1:2010(E) : *Information technology – Programming Languages – Fortran; Part 1: Base Language*

2 Requirements

2.1 General

The subclauses of this clause contain a general description of the extensions to the syntax and semantics of the Fortran programming language to provide abstract subprograms, to instantiate them, to use them to specify explicit interfaces, and to invoke instantiations of them.

2.2 Summary

2.2.1 What is provided

This technical specification defines a new form of subprogram definition, called an abstract subprogram. An abstract subprogram is a definition of a family of programs. An abstract subprogram cannot be invoked. Instead, one can instantiate a member of that family, or specify an explicit interface, by providing values for parameters using integer constant expressions.

This technical specification defines mechanisms to cause instantiations of abstract subprograms to be created. An instantiation of an abstract subprogram is a subprogram that behaves in all respects but one in exactly the same ways as a subprogram defined by International Standard ISO/IEC 1539-1:2010(E). The single exception is that an instantiation of an abstract subprogram does not access the scoping unit containing its instantiation by host association; rather, it accesses the scoping unit containing the definition of the abstract subprogram of which it is an instantiation by host association.

This technical specification defines mechanisms by which abstract subprograms can be used to specify explicit interfaces, by providing values for parameters using integer constant expressions.

2.2.2 Abstract subprogram

An abstract subprogram is a definition of a family of subprograms, characterized by integer parameters.

2.2.3 Instantiation of an abstract subprogram

An instantiation of an abstract subprogram is a member of the family of subprograms defined by the referenced abstract subprogram. It is characterized by integer constant expressions, and behaves in all respects but one in exactly the same ways as a subprogram defined by International Standard ISO/IEC 1539-1:2010(E). The single exception is that an instantiation of an abstract subprogram does not access the scoping unit containing its instantiation by host association; rather, it accesses the scoping unit containing the definition of the abstract subprogram of which it is an instantiation by host association. The only case where this distinction has effect is where an abstract subprogram is defined in a module, and instantiated in a different scoping unit; in all other cases, instantiations of an abstract subprogram can only be created in the same scoping unit as the abstract subprogram.

2.2.4 Explicit interface specified using an abstract subprogram

An instantiation of an abstract subprogram has explicit interface. An explicit interface can be specified, using an abstract subprogram and values for its parameters, without instantiating it, if the name being declared has the POINTER attribute or is a dummy argument.

2.3 Syntax to define an abstract subprogram

An abstract subprogram is a subprogram defined using the facilities for subprogram definition provided by International Standard ISO/IEC 1539-1:2010(E), and including in addition the word ABSTRACT,

following by a parenthesized list of names and optional default values for parameters, in the *prefix* of its initial statement.

The definition of *prefix-spec* is revised:

R1226 *prefix-spec* **is** *declaration-type-spec*
or ABSTRACT (*parameter-name-list*)
or ELEMENTAL
or IMPURE
or MODULE
or PURE
or RECURSIVE

The procedure parameter definition statement is introduced:

R1226a *subprogram-param-def-stmt* **is** INTEGER, KIND :: *subprogram-param-def-list*
R1226b *subprogram-param-def* **is** *parameter-name* [= *scalar-int-constant-expr*]

2.4 Syntax to instantiate an abstract subprogram

An instantiation of an abstract subprogram is directly created by a *procedure-stmt* or a *procedure-declaration-stmt*. A requirement to instantiate an abstract subprogram, depending upon the declaration of an object, is specified by a *type-bound-procedure-stmt* of a *final-procedure-stmt*.

The definition of *type-bound-procedure-stmt* is revised:

R448 *type-bound-procedure-stmt* **is** PROCEDURE [[, *binding-attr-list*] ::] ■
■ *type-bound-proc-decl-list*
or PROCEDURE (*interface-name*), ■
■ *binding-attr-list* :: *binding-name-list*
or PROCEDURE (*abstract-subprogram-ref*), ■
■ *binding-attr-list* :: *binding-name*

Constraint C477 is revised:

C477 (R451) DEFERRED shall appear if *interface-name* appears. DEFERRED shall not appear if neither *interface-name* nor *abstract-subprogram-ref* appears.

The definition of *final-procedure-stmt* is revised:

R452 *final-procedure-stmt* **is** FINAL [::] *final-subprogram-name-list*
or FINAL (*abstract-subprogram-ref*)

The definition of *procedure-stmt* is revised:

R1206 *procedure-stmt* **is** [MODULE] PROCEDURE [::] *procedure-name-list*
or PROCEDURE (*abstract-subprogram-ref*) [::] ■
■ *procedure-name*

The definition of *procedure-declaration-stmt* is revised:

R1211 *procedure-declaration-stmt* **is** PROCEDURE ([*proc-interface*]) ■
■ [[, *proc-attr-spec*] ... ::] *proc-decl*-list
or PROCEDURE (*abstract-subprogram-ref*) ■
■ [[, *proc-attr-spec*] ... ::] *proc-decl*

The definition of *abstract-subprogram-ref* is introduced:

R1211a *abstract-subprogram-ref* **is** *abstract-subprogram-name* (*parameter-spec-list*)

The definition of *parameter-spec* is introduced:

R1211b *parameter-spec* **is** [*parameter-name* =] *scalar-int-constant-expr*

C1215a (R1211a) The *abstract-subprogram-name* shall be the name of an abstract subprogram.

C1215b (R1211b) The *parameter-name* = may be omitted from a *parameter-spec* only if the *parameter-name* = has been omitted from each preceding *parameter-spec* in the *parameter-spec-list*.

C1215c (R1211b) Each *parameter-name* shall appear in the *parameter-name-list* of the abstract subprogram.

C1215d (R1211a) A *parameter-spec* shall be provided for each *parameter-name* of the abstract subprogram for which a default value is not specified.

2.5 Syntax to use an abstract subprogram to specify an explicit interface

An abstract subprogram definition can be used to specify an explicit interface by including values for its parameters.

The definition of *proc-component-def-stmt* is revised:

R440 *proc-component-def-stmt* **is** PROCEDURE ([*proc-interface*) , ■
■ *proc-component-attr-spec-list* :: *proc-decl-list*
or PROCEDURE ([*abstract-subprogram-ref*) ■
■ *proc-component-attr-spec-list* :: *proc-decl*

If the *procedure-entity-name* in a *proc-decl* in a *procedure-declaration-stmt* has the POINTER attribute, or if the *procedure-entity-name* is the name of a dummy procedure, the *abstract-subprogram-ref* specifies an explicit interface for the *procedure-entity-name*.

If the *binding-name* in a *type-bound-procedure-stmt* has the DEFERRED attribute, the *abstract-subprogram-ref* specifies an explicit interface for the *binding-name*.

2.6 Definition of an abstract subprogram

An abstract subprogram is defined within the *specification-part* of a main program, module, external subprogram, or module subprogram, by a *function-subprogram* or *subroutine-subprogram* that has ABSTRACT (*parameter-name-list*) in its initial statement.

An abstract subprogram shall not contain an ENTRY statement.

2.7 Instantiation of an abstract subprogram

Direct instantiation of an abstract subprogram occurs where a *procedure-stmt* appears with *abstract-subprogram-ref*, provided the *procedure-name* is not the name of a dummy procedure. The name of the instantiation is *procedure-name*.

Direct instantiation of an abstract subprogram occurs where a *procedure-declaration-stmt* appears with *abstract-subprogram-ref* and the declared *procedure-entity-name* is not the name of a dummy procedure and does not have the POINTER attribute. The name of the instantiation is *procedure-entity-name*.

Indirect instantiation of an abstract subprogram occurs where an object of a derived type is declared, providing it is not a dummy argument, and the definition of the type of the object includes a *type-bound-procedure-stmt* with *abstract-subprogram-ref* and without the DEFERRED attribute, or a *final-procedure-stmt* with *abstract-subprogram-ref*. An indirect instantiation does not have a name, but is bound to the *binding-name* in the case of a *type-bound-procedure-stmt*.

Instantiation of an abstract subprogram causes each appearance of a *parameter-name* within the abstract subprogram to be replaced in the instantiation by the value of the corresponding *scalar-int-constant-expr* in the *abstract-subprogram-ref*, if one appears, or by the *scalar-int-constant-expr* immediately following *parameter-name* = in the *subprogram-param-def-stmt* otherwise. Each *parameter-spec* in an *abstract-subprogram-ref* that does not include *parameter-name* corresponds to the *parameter-name* in the same position in the *parameter-name-list* of the abstract subprogram. Each *parameter-spec* that includes *parameter-name* corresponds to the *parameter-name* in the *parameter-name-list* that has the same *parameter-name*. There shall not be more than one *parameter-spec* corresponding to each *parameter-name*. There shall be a *parameter-spec* corresponding to each *parameter-name* for which a default value is not specified.

An abstract subprogram shall not be instantiated, directly or indirectly, within the inclusive scoping unit of an internal subprogram. If it is instantiated within the inclusive scoping unit of a main program, external subprogram, or module subprogram, including within a BLOCK construct, the instantiation is an internal subprogram of that inclusive scoping unit. If it is instantiated within a BLOCK construct, the name of the instantiation has a scope of the construct. If it is instantiated within a module, the instantiation is a module subprogram.

2.8 Invoking an instantiation of an abstract subprogram

An instantiation of an abstract subprogram is invoked by a *function-reference* or *call-stmt*. If it is a direct instantiation, the name specified in the instantiation is used as the *procedure-designator*. If it is an indirect instantiation, its binding name is used as the *procedure-designator*. If an instantiation is a final procedure, it is invoked according to the rules in subclause 4.5.6.2 of ISO/IEC 1539-1:2010(E).

2.9 Constant expression

The definition of constant expression is expanded to encompass the use within it of a *parameter-name* within an abstract subprogram.

Item (9a) is added to the list of primaries allowed in a constant expression:

(9a) a previously-declared *parameter-name* of the abstract subprogram being defined,

2.10 Scoping units and host association

An abstract subprogram is a scoping unit. It accesses the scoping unit in which it is defined by host association. An instantiation of it does not access, by host association, the scoping unit in which it is instantiated. The only case where this distinction has effect is where the definition appears in a module. In the cases of the definition appearing in a main program, external subprogram, or module subprogram, instantiation cannot occur in any other inclusive scoping unit.

3 Examples

3.1 Definition of an abstract subprogram

```
pure abstract ( RK ) function Planck ( Frequency, Temperature )
  integer, kind :: RK
  real(rk) :: Planck
  real(rk), intent(in) :: Frequency     ! MHz
  real(rk), intent(in) :: Temperature   ! Kelvin
  real(rk), parameter :: H = 6.62606947e-34_rk       ! J s, +/- 29e-42 NIST 2010
  real(rk), parameter :: K = 1.3806488e-23_rk        ! J/K, +/- 13e-30 NIST 2010
  real(rk), parameter :: H_OVER_K = H / K * 1.0e6_rk ! nu in MHz
  real(rk) :: A, R, HXF
  hxf = h_over_k * frequency
  r = hxf / temperature
  a = exp(r) - 1.0
  planck = hxf / a
end function Planck
```

3.2 Direct instantiation of an abstract subprogram

```
interface Planck
  procedure(Planck(kind(0.0e0))) :: Planck_single
  procedure(Planck(kind(0.0d0))) :: Planck_double
end interface Planck
```

3.3 Indirect instantiation of an abstract subprogram

```
type :: Rad_Tran ( RK )
  integer, kind :: RK
  real(rk) :: Radiance
contains
  procedure(Planck(rk))
end type Rad_Tran

integer, parameter :: Q = selected_real_kind(30)

type(Rad_Tran(q)) :: Rad_Q
```

3.4 Reference to directly instantiated abstract subprogram

```
  print *, Planck ( 1.42857d+4, 2.30d0 ) ! MHz, Kelvin
```

3.5 Reference to indirectly instantiated abstract subprogram

```
  rad_q%radiance = rad_q%planck ( 1.42857e+4_q, 2.30e0_q ) ! MHz, Kelvin
```


4 Required editorial changes to ISO/IEC 1539-1:2010(E)

The following editorial changes to ISO/IEC 1539-1:2010(E), if implemented, would provide the facilities described in foregoing clauses of this technical specification. Descriptions of how and where to place the new material are enclosed between square brackets.

[Introduction:xv] Add an item to the list of new features:

- Procedure enhancements:
  Abstract subprograms define families of subprograms parameterized by integer constant expressions. Specific instantiations of them can be created, and they can be used to provide explicit interfaces.

[1.3.1+ 2:10+] Insert term definitions:

1.3.1a
abstract subprogram
definition of a family of subprograms, characterized by integer parameters

1.3.1b
abstract subprogram instantiation
subprogram created by reference to an abstract subprogram, with values specified for its parameters

---

[2.1 27:28+] Introduce definition of *abstract-subprogram*:

R1236a *abstract-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*

[2.1 R207 28:11] Introduce a new first alternative of *declaration-construct*:

R207 *declaration-construct* **is** *abstract-subprogram*
or *derived-type-def*

[2.1 R207 28:18+] Connect *subprogram-param-def-stmt* to *specification-part*:

or *subprogram-param-def-stmt*

[4.5.4.1 R440 67:4+] Add an alternative to *proc-component-def-stmt*:

or PROCEDURE (*abstract-subprogram-ref*) ■
■ *proc-component-attr-spec-list* :: *proc-decl*

[4.5.5 R448 73:12+] Add an alternative to *type-bound-procedure-stmt*:

or PROCEDURE (*abstract-subprogram-ref*), ■
■ *binding-attr-list* :: *binding-name*

[4.5.5 74:8] Revise C477:

C477 (R451) DEFERRED shall appear if *interface-name* appears. DEFERRED shall not appear if neither *interface-name* nor *abstract-subprogram-ref* appears.

[4.5.6.1 R452 75:8+] Add an alternative to *final-procedure-stmt*:

or FINAL (*abstract-subprogram-ref*)

[7.1.12p1(9)+ 152:9+] Make a parameter of an abstract procedure a constant expression primary:

(9a) a previously-declared *parameter-name* of the abstract subprogram being defined,

---

[12.2.2.2p3 277:27] Specify that an instantiation of an abstract subprogram within a module defines a module procedure. At the end of the sentence append "or by instantiation within a module of an abstract subprogram".

[12.4.2.1p1 279:17] Append a sentence: "An interface declared by an *abstract-subprogram-ref* is explicit."

[12.4.3.2 R1206 280:22+] Add an alternative to *procedure-stmt*:

or PROCEDURE (*abstract-subprogram-ref*) [::] ■
■ *procedure-name*

[12.4.3.4.1 283:12] Allow generic name to be the same as an abstract subprogram name. After "interface block" insert ", an abstract subprogram name".

[12.4.3.6 R1211+ 287:8+] Add an alternative to *procedure-declaration-stmt*:

or PROCEDURE (*abstract-subprogram-ref*) ■
■ [[, *proc-attr-spec*] ... ::] *proc-decl*

Introduce definition of *abstract-subprogram-ref*:

R1211a *abstract-subprogram-ref* **is** *abstract-subprogram-name* (*parameter-spec-list*)

Introduce definition of *parameter-spec*:

R1211b *parameter-spec* **is** [*parameter-name* =] *scalar-int-constant-expr*

[12.4.3.6 287:22+] Introduce constraints on *abstract-subprogram-ref* and *parameter-spec*:

C1215a (R1211a) The *abstract-subprogram-name* shall be the name of an abstract subprogram.

C1215b (R1211b) The *parameter-name* = may be omitted from a *parameter-spec* only if the *parameter-name* = has been omitted from each preceding *parameter-spec* in the *parameter-spec-list*.

C1215c (R1211b) Each *parameter-name* shall be a parameter name specified in the *parameter-name-list* of the abstract subprogram.

C1215d (R1211a) A *parameter-spec* shall be provided for each *parameter-name* of the abstract subprogram for which a default value is not specified.

[12.6.2.1 R1226 305:25+] Introduce an alternative to *prefix-spec*:

or ABSTRACT (*parameter-name-list*)

[12.6.2.1 305:30+] Introduce definition of *subprogram-param-def-stmt*:

R1226a *subprogram-param-def-stmt* **is** INTEGER, KIND :: *subprogram-param-def-list*
R1226b *subprogram-param-def* **is** *parameter-name* [= *scalar-int-constant-expr*]

[12.6.2.1 305:34+] Require declaration of parameter names within the scoping unit of the abstract sub-

program:

C1246a (R1225) Every *parameter-name* shall appear in a *subprogram-param-def-stmt* within the scoping unit of the abstract procedure being defined.

C1246b (R1226a) A *subprogram-param-def-stmt* shall not appear except within the scoping unit of an abstract subprogram.

C1246c (R1226b) The *parameter-name* shall be a parameter name of the abstract procedure being defined.

[12.6.2.1 306:6+] Prohibit separating the interface and implementation of abstract procedures:

C1251a (R1225) If ABSTRACT appears, MODULE shall not appear.

[12.6.2.1 306:12+] Define abstract subprogram parameter defaults. Insert a paragraph:

A *scalar-int-constant-expr* in a *subprogram-param-def* specifies a default value for a *parameter-name*, which is used within an instantiation of the abstract subprogram if and only if a *scalar-int-constant-expr* is not specified in the *abstract-subprogram-ref* of the instantiation.

[12.6.2.3+ 308:17+] Introduce subclauses:

12.6.2.3a Abstract subprogram

An abstract subprogram is a subprogram defined by a *function-subprogram* or *subroutine-subprogram* in which ABSTRACT (*parameter-decl-list*) appears in its *function-stmt* or *subroutine-stmt*, respectively. It defines a family of subprograms characterized by integer parameters.

R1236a *abstract-subprogram* **is** *function-subprogram*
or *subroutine-subprogram*

C1261a (R1236a) ABSTRACT (*parameter-decl-list*) shall appear in the *function-stmt* or *subroutine-stmt* of the *function-subprogram* or *subroutine-subprogram*.

A parameter of an abstract subprogram is a constant.

12.6.2.3b Instantiation of an abstract subprogram

Except as specified in subclause 12.6.2.3c, an abstract subprogram is instantiated directly by a *procedure-stmt* or *procedure-declaration-stmt* in which *abstract-subprogram-ref* appears. The name of the instantiation is the *procedure-name* specified in the *procedure-stmt* or *procedure-declaration-stmt*. Each direct instantiation of an abstract subprogram is a different subprogram.

Except as specified in subclause 12.6.2.3c, where an object of derived type is declared, an abstract subprogram is indirectly instantiated for each binding name in each *type-bound-procedure-stmt* that includes *abstract-subprogram-ref*, and each FINAL statement that includes *abstract-subprogram-ref*, in the definition of its type. An indirect instantiation has no name, but if it results from a *type-bound-procedure-stmt*, it is bound to the type using the *binding-name*. Each instantiation resulting from a FINAL statement that includes *abstract-subprogram-ref* is bound to the type as a final subroutine. It is processor dependent whether indirect instantiations of an abstract subprogram with identical subprogram parameter values are the same or different subprograms.

NOTE 12.43a

Different instantiations do not share local variables that have the SAVE attribute. If an abstract subprogram has no local SAVE variables, a program cannot detect whether indirect instantiations

NOTE 12.43a (cont.)

are the same or different subprograms.

Unresolved Technical Issue SAVE

An alternative is to prohibit indirect instantiation of subprograms that have SAVE variables.

An instantiation is created by replacing each appearance of *parameter-name* within the abstract subprogram by the *scalar-int-constant-expr* in the *abstract-subprogram-ref* that corresponds to the *parameter-name* in the abstract subprogram definition, if there is a corresponding expression, or by the default value for the *parameter-name* otherwise. An instantiation of an abstract subprogram has an explicit interface. An instantiation of an abstract subprogram has all the attributes specified in the *prefix* of the abstract subprogram, except the ABSTRACT attribute.

A *parameter-spec* in an *abstract-subprogram-ref* corresponds to a *parameter-decl* in the same position in the *parameter-decl-list* in the *function-stmt* or *subroutine-stmt* of the specified abstract subprogram if it does not include a parameter name, and otherwise corresponds to the parameter of the abstract subprogram that has the same name.

C1261b An abstract subprogram shall not be instantiated except within a main program, module, module subprogram, or external subprogram.

12.6.2.3c Abstract subprogram references that do not cause instantiation

An instantiation of an abstract subprogram is not created for

- a derived type component declared by a *proc-component-def-stmt*,
- a binding declared by a *type-bound-procedure-stmt* that specifies the DEFERRED attribute,
- a dummy procedure declared in a *procedure-stmt* or *procedure-declaration-stmt*,
- a procedure pointer,
- a *procedure-stmt* or *procedure-declaration-stmt* within an interface body, or
- a binding to an object of derived type if the object
  - is a dummy argument,
  - is declared within an interface body,
  - is declared within the specification part of a module or submodule and has the POINTER or ALLOCATABLE attribute, or
  - is declared within the specification part of a subprogram or BLOCK construct, has the POINTER or ALLOCATABLE attribute, and the object does not appear as an *allocate-object* in an ALLOCATE statement within the inclusive scoping unit of the declaration, or an inclusive scoping unit that accesses that scoping unit by host association.

[12.6.2.6 C1265 309:33] Although redundant, because an abstract subprogram is neither an *external-subprogram* nor a *module-subprogram*, append "or an *abstract-subprogram*" at the end of C1265.

[16.3.1p1(1)+ 440:5] Specify that the identifier of an abstract subprogram and the identifiers of its parameters are local identifiers: After "statement functions" insert ", abstract subprograms, abstract subprogram parameters".

Define abstract subprogram parameter keywords

[15.3.5+ 442:10+] Introduce a subclause:

16.3.6 Abstract subprogram parameter keywords

As an abstract subprogram parameter keyword, an abstract subprogram parameter name has a scope of the scoping unit of the host of the abstract subprogram definition. It may appear as an abstract subprogram parameter keyword only in an *abstract-subprogram-ref* for the subprogram of which it is a parameter. If the abstract subprogram definition is accessible in another scoping unit by use or host association (16.5.1.3, 16.5.1.4), the abstract subprogram parameter keyword is accessible for subprogram instantiations for that abstract subprogram in that scoping unit.

Specify that abstract subprograms and instantiations of them access, by host association, the scoping unit in which the abstract subprogram definition appears.

[16.5.1.4p1 443:28] Replace "An instance" by "Except for instantiations of abstract subprograms, an instance"

[16.5.1.4p1 443:29] Between the first and second sentences, insert a sentence

"An abstract subprogram definition and instantiations of it access the host of the abstract subprogram definition by host association."

[16.5.1.4p2(3)+ 444:7+] Specify that an abstract subprogram name is a local identifier:

(2a) A *subroutine-name* or a *function-name* in the *function-stmt* or *subroutine-stmt* of an *abstract-subprogram*,

---

[16.5.1.4p2(3)+ 444:8+] Specify that an abstract subprogram parameter name is a local identifier:

(3a) A *parameter-name* in a *subprogram-param-def-stmt*,

[16.5.1.4p2(3)+ 444:26] Specify that an abstract subprogram name is a local identifier of the scoping unit in which it is defined. Before "Local identifiers" insert "A *subroutine-name* or a *function-name* in the *function-stmt* or *subroutine-stmt* of an *abstract-subprogram* is a local identifier in the scoping unit where the abstract subprogram definition appears; any entity of the host that has this as its nongeneric name is inaccessible by that name."

[A.2 462:1+] Insert a list item

- whether different indirect instantiations of an abstract subprogram with identical subprogram parameter values are the same subprogram (12.6.2.3b);