Programming languages — Ada

DEFECT REPORTS

Part 1

For ISO/IEC 8652:1995

September 2000

This document was prepared by AXE Consulting under contract from The MITRE Corporation.

Introduction

This document contains defect reports on the Ada 95 standard [ISO/IEC 8652:1995], and responses formulated by the Ada Rapporteur Group (ARG) of ISO/IEC JTC 1/SC 22/WG 9, the Ada working group. The ARG is the language maintenance subgroup of WG 9, meaning that it is responsible for determining the corrections to the standard.

Defect Reports usually come from comments submitted by the public to the ARG. These comments are distilled into a question, given in the **Question** section of the Defect Report response.

In order to formulate the response to the Defect Report, the question is carefully considered and often discussed at length by the ARG. The results are recorded in the **Discussion** section of the response. The answer to the question arrived at after the discussions is summarized in the **Summary of Response** section. A more detailed answer to the question can be found in the **Response** section. Sometimes, the issue is so obvious that there is no **Response** or **Discussion** section. If the result of the discussion finds that some change to the standard would be required to arrive at an answer to the question, a **Corrigendum Wording** section includes the specific wording change to standard. These **Corrigendum Wording** sections are gathered together in a Technical Corrigendum document.

A Defect Report and Response is the final step of a lengthy process of formulation, discussion, and approval. The working documents of the ARG (called Ada Issues) contain additional information about the issue and its resolution. Ada Issues may include sections for testing information (**ACATS test**), informal wording changes (**Wording**), and an appendix including E-Mail comments on this issue (**Appendix**). These sections are not included in the Defect Reports found in this document. This information is available in the Ada Issues documents, which can be accessed on the web at www.ada-auth.org/~acats/arg.

The Defect Reports and Responses contain many references of the form ss.cc(pp) or ss.cc.aa(pp). These refer to particular paragraphs in the standard, with the notation referencing the (sub)clause number in the Ada 95 standard (ss.cc.aa), followed by a parenthesized paragraph number (pp). Paragraphs are numbered by counting from the top of the (sub)clause, ignoring headings.

The Defect Reports and Responses contain references to the Annotated Ada Reference Manual (AARM). This document contains all of the text in the Ada 95 standard along with various annotations. It was prepared by the Ada 95 design team, and is intended primarily for compiler writers, test writers, and the ARG. The annotations include rationale for some rules. The AARM is often used by the ARG to determine the intent of the language designers.

The Defect Reports and Responses may contain references to Ada 83. Ada 83 is the common name for the previous version of the Ada standard, ISO/IEC 8652:1987. Similarly, AI83 refers to interpretations of that standard.

This document contains all of the Defect Reports used to prepare Ada Technical Corrigendum 1. Issues which did not result in wording changes to the standard are available in the companion document, Defect Reports Part 2. Resolutions of newer issues can be found on the web site mentioned previously.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0001**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **1.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0001 The AE characters are allowed in identifiers

Working Reference Number AI95-00124

Question

2.1(8-9) say:

upper_case_identifier_letter

Any character of Row 00 of ISO 10646 BMP whose name begins ‘‘Latin Capital Letter’’.

lower_case_identifier_letter

Any character of Row 00 of ISO 10646 BMP whose name begins ‘‘Latin Small Letter’’.

The letters allowed in identifiers are then restricted to lower_case_identifier_letters and upper_case_identifier_letters.

The version of 10646-1:1993 referred to in 1.2(8) names codes C6 and E6 as "Latin Capital Ligature AE" and "Latin Small Ligature AE".

This seems to imply that these characters are not allowed in identifiers. Are these characters allowed in identifiers? (Yes.)

Summary of Response

The characters LATIN CAPITAL LETTER AE and LATIN SMALL LETTER AE are allowed in identifiers.

Corrigendum Wording

Replace 1.2(8):

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*.

by:

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*, supplemented by Technical Corrigendum 1:1996.

Discussion

Technical Corrigendum 1 of 10646 names these characters LATIN CAPITAL LETTER AE and LATIN SMALL LETTER AE. The intent was that these letters be allowed in identifiers.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0002**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.3.1; 3.6; 3.8; 4.8; 9.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0002 Elaboration of subtype_indications with per-object constraints

Working Reference Number AI95-00171

Question

When does the elaboration of a subtype indication with a per-object constraint occur? What are the actions of such an elaboration?

When a component has a subtype_indication with a per-object constraint and an object of the type containing the component is declared, the subtype_indication containing the per-object constraint is apparently never elaborated.

3.8(18) explains that subtype_indications with per-object constraints are not elaborated, but that any expressions that are not part of a per-object expression are evaluated. However, what to do with the results of those evaluations never seems to be explained.

3.3.1(15-20) describes the process of elaborating an object_declaration. In step 3, per-object expressions are evaluated, but there is no mention of elaborating anything, although later paragraph 20 does seem to imply that some sort of elaborations were supposed to have taken place in step 3.

The elaboration of per-object constraints is mentioned in (at least) the following other places where objects are created:

4.3.1(19) creating record aggregates

4.8(10) creating heap objects via uninitialized allocators

9.4(14) creating a protected object (This one is supposed to be redundant with 3.3.1, but in fact the two appear to be inconsistent.)

According to a strict reading, elaborating the per-object constraint would appear to involve reevaluating the non-per-object expressions (since there doesn't seem to be any separate definition of what happens when a per-object constraint is elaborated), but not include any subtype compatibility checks that would normally occur as part of subtype elaboration (since elaboration of the subtype_indication containing the constraint isn't mentioned in these paragraphs). What are the intended semantics?

Summary of Response

The elaboration of a subtype indication with a per-object constraint occurs when an object of the enclosing type is created. This elaboration consists of the evaluation of each per-object expression of the constraint, followed by the usual actions associated with such elaboration, but using the values for any expressions that are not part of a per-object expression that were determined earlier when the type definition was elaborated.

For evaluating a named association applying to multiple components in a per-object discriminant constraint, if the expression of the association is not part of a per-object expression, then it must be evaluated once for each associated component.

Corrigendum Wording

Replace 3.3.1(18):

3. The object is created, and, if there is not an initialization expression, any per-object expressions (see 3.8) are evaluated and any implicit initial values for the object or for its subcomponents are obtained as determined by the nominal subtype.

by:

3. The object is created, and, if there is not an initialization expression, any per-object constraints (see 3.8) are elaborated and any implicit initial values for the object or for its subcomponents are obtained as determined by the nominal subtype.

Replace 3.6(22):

The elaboration of a discrete_subtype_definition creates the discrete subtype, and consists of the elaboration of the subtype_indication or the evaluation of the range. The elaboration of a component_definition in an array_type_definition consists of the elaboration of the subtype_indication. The elaboration of any discrete_subtype_definitions and the elaboration of the component_definition are performed in an arbitrary order.

by:

The elaboration of a discrete_subtype_definition that does not contain any per-object expressions creates the discrete subtype, and consists of the elaboration of the subtype_indication or the evaluation of the range. The elaboration of a discrete_subtype_definition that contains one or more per-object expressions is defined in 3.8. The elaboration of a component_definition in an array_type_definition consists of the elaboration of the subtype_indication. The elaboration of any discrete_subtype_definitions and the elaboration of the component_definition are performed in an arbitrary order.

Replace 3.8(18):

Within the definition of a composite type, if a component_definition or discrete_subtype_definition (see 9.5.2) includes a name that denotes a discriminant of the type, or that is an attribute_reference whose prefix denotes the current instance of the type, the expression containing the name is called a *per-object expression*, and the constraint being defined is called a *per-object constraint*. For the elaboration of a component_definition of a component_declaration, if the constraint of the subtype_indication is not a per-object constraint, then the subtype_indication is elaborated. On the other hand, if the constraint is a per-object constraint, then the elaboration consists of the evaluation of any included expression that is not part of a per-object expression.

by:

Within the definition of a composite type, if a component_definition or discrete_subtype_definition (see 9.5.2) includes a name that denotes a discriminant of the type, or that is an attribute_reference whose prefix denotes the current instance of the type, the expression containing the name is called a *per-object expression*, and the constraint or range being defined is called a *per-object constraint*. For the elaboration of a component_definition of a component_declaration or the discrete_subtype_definition of an entry_declaration for an entry family (see 9.5.2), if the constraint or range of the subtype_indication or discrete_subtype_definition is not a per-object constraint, then the subtype_indication or discrete_subtype_definition is elaborated. On the other hand, if the constraint or range is a per-object constraint, then the elaboration consists of the evaluation of any included expression that is not part of a per-object expression. Each such expression is evaluated once unless it is part of a named association in a discriminant constraint, in which case it is evaluated once for each associated discriminant.

When a per-object constraint is elaborated (as part of creating an object), each per-object expression of the constraint is evaluated. For other expressions, the values determined during the elaboration of the component_definition or entry_declaration are used. Any checks associated with the enclosing subtype_indication or discrete_subtype_definition are performed, including the subtype compatibility check (see 3.2.2), and the associated subtype is created.

Replace 4.8(10):

- If the designated type is composite, an object of the designated type is created with tag, if any, determined by the subtype_mark of the subtype_indication; any per-object constraints on subcomponents are elaborated and any implicit initial values for the subcomponents of the object are obtained as determined by the subtype_indication and assigned to the corresponding subcomponents. A check is made that the value of the object belongs to the designated subtype. Constraint_Error is raised if this check fails. This check and the initialization of the object are performed in an arbitrary order.

by:

- If the designated type is composite, an object of the designated type is created with tag, if any, determined by the subtype_mark of the subtype_indication; any per-object constraints on subcomponents are elaborated (see 3.8) and any implicit initial values for the subcomponents of the object are obtained as determined by the subtype_indication and assigned to the corresponding subcomponents. A check is made that the value of the object belongs to the designated subtype. Constraint_Error is raised if this check fails. This check and the initialization of the object are performed in an arbitrary order.

Replace 9.5.2(22):

For the elaboration of an entry_declaration for an entry family, if the discrete_subtype_definition contains no per-object expressions (see 3.8), then the discrete_subtype_definition is elaborated. Otherwise, the elaboration of the entry_declaration consists of the evaluation of any expression of the discrete_subtype_definition that is not a per-object expression (or part of one). The elaboration of an entry_declaration for a single entry has no effect.

by:

The elaboration of an entry_declaration for an entry family consists of the elaboration of the discrete_subtype_definition, as described in 3.8. The elaboration of an entry_declaration for a single entry has no effect.

Response

Notwithstanding the rules given in 3.3.1(18), 4.3.1(19), and 4.8(10), the elaboration of the subtype indication of a component definition with a per-object constraint occurs when an object of the enclosing type is created. This elaboration takes place on elaboration of an object declaration, evaluation of an uninitialized allocator, and when evaluating an aggregate of the type.

The elaboration consists of the evaluation of each per-object expression of the component's constraint, followed by the conversion of the value of each expression of the constraint to its appropriate expected type and the performance of the compatibility check defined for the elaboration of the subtype indication (see 3.2.2(11)). The values used for any expressions that are not part of per-object expressions of the subtype's constraint are those determined during the original elaboration of the component definition as defined in 3.8(18). Such expressions are not reevaluated during elaboration of the per-object constraint that occurs as part of object creation, despite any rules that state when a per-object constraint is elaborated (e.g., as part of evaluating an allocator or aggregate).

Note further that the evaluation of expressions in a per-object constraint defined in 3.8(18) was intended to take into account the case of named associations for multiple components in a discriminant constraint. For such an association, the expression must be evaluated once for each associated component, as prescribed by 3.7.1(12).

Discussion

There are two basic problems with the current wording of the standard regarding the elaboration of components with a per-object constraint. The first is that the rules don't explain what is done with the values obtained from expressions that are not part of per-object expressions (as defined in 3.8(18)) or whether such expression are reevaluated when a per-object constraint is later elaborated during object creation. The other problem is that the mention of elaboration of per-object constraints in rules such as 4.3.1(19) and 4.8(10) fails to cover the need for the subtype compatibility check that is normally performed when elaborating a subtype indication.

The intent was clearly that the values of the expressions evaluated as part of elaborating a component definition with a per-object constraint (3.8(18)) should be used later when creating an object of the containing type. It would not make sense to discard the values already determined and to reevaluate the expressions (especially if they have side effects). The description in the rules for allocator and aggregate evaluation that states that a per-object constraint is elaborated should mention that only the per-object expressions are evaluated at that point and that the values for other expressions are those determined earlier

when the type was elaborated. (The description of the semantics of elaborating per-object constraints should really be centralized in a single place, such as 3.8(18).)

The rules for object declarations, allocator evaluation, and aggregate evaluation all fail to require the subtype compatibility check that occurs when a subtype indication is elaborated (and for object declarations even the constraint elaboration is omitted). This check is certainly needed in these cases as well. The fix for this oversight is to define each of these rules to include the elaboration of the subtype indications for components with per-object constraints (which also subsumes the elaboration of the constraint itself).

One other minor gap is that the case of elaborating a named discriminant association within a per-object constraint is not covered by that rule in 3.8(18). The rule as given only describes a single evaluation for each expression of the constraint, but the intent is that for a named association the expression should be evaluated for each associated component.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0003**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **3.5.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0003 Modular types on one's complement machines

Working Reference Number AI95-00095

Question

How should an implementation on a one's complement machine implement modular types intended to use all the bits of a full word?

Summary of Response

Implementation Permission: On a one's complement machine, the implementation may support non-binary moduli above System.Max_Nonbinary_Modulus.

Corrigendum Wording

Insert after 3.5.4(27):

> For a one's complement machine, the high bound of the base range of a modular type whose modulus is one less than a power of 2 may be equal to the modulus, rather than one less than the modulus. It is implementation defined for which powers of 2, if any, this permission is exercised.

the new paragraph:

> For a one's complement machine, implementations may support non-binary modulus values greater than System.Max_Nonbinary_Modulus. It is implementation defined which specific values greater than System.Max_Nonbinary_Modulus, if any, are supported.

Response

Consider a 36-bit one's complement machine. One should be able to declare a 36-bit modular type. For logical operations to make sense, the all-ones bit pattern ought to be allowed, and should compare not equal to zero, and greater than every other bit pattern. The Implementation Permission in 3.5.4(27) is intended to allow this.

On a 36-bit two's complement machine, one would declare:

```
type T is mod 2**36;
```

and T'Modulus would be 2**36, and the base range of T would be 0..2**36-1. If one says:

```
type TT is mod 2**36-1;
```

TT'Modulus is 2**36-1, and the base range of TT is usually 0..2**36-2. The implementation permission says that the base range of TT can be 0..2**36-1. This means that the all-ones bit pattern is a valid value of the type, and is not reduced via the modulus.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0004**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.5.8**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0004 S'Digits when T'Machine_Radix is 10

Working Reference Number AI95-00203

Question

The relationship given in 3.5.8(2) in the case of T'Machine_Radix = 10 implies that S'Digits + 1 = T'Model_Mantissa in such a case. Is this correct? (No.)

Summary of Response

The relationship between S'Digits and T'Model_Mantissa given in 3.5.8(2) states that S'Digits is the largest value of d for which

ceiling(d * log(10) / log(T'Machine_Radix)) + 1 <= T'Model_Mantissa

This allows for a "guard digit" which is necessary to take care of extreme circumstances that arise if the Machine_Radix is not decimal (as is usually the case).

However, this guard digit is unnecessary if Machine_Radix is 10 or a power of 10 and in such a case the relationship should read

ceiling(d * log(10) / log(T'Machine_Radix)) <= T'Model_Mantissa

If Machine_Radix is 10 this becomes simply

d <= T'Model_Mantissa

so that S'Digits = T'Model_Mantissa.

Corrigendum Wording

Replace 3.5.8(2):

S'Digits S'Digits denotes the requested decimal precision for the subtype S. The value of this attribute is of the type *universal_integer*. The requested decimal precision of the base subtype of a floating point type T is defined to be the largest value of *d* for which ceiling(*d* * log(10) / log(T'Machine_Radix)) + 1 <= T'Model_Mantissa.

by:

S'Digits S'Digits denotes the requested decimal precision for the subtype S. The value of this attribute is of the type *universal_integer*. The requested decimal precision of the base subtype of a floating point type T is defined to be the largest value of *d* for which

ceiling(*d* * log(10) / log(T'Machine_Radix)) + *g* <= T'Model_Mantissa

where *g* is 0 if Machine_Radix is a positive power of 10 and 1 otherwise.

Discussion

This question echoes back to a change made between 1980 preliminary Ada and the 1983 standard which is worth explaining as background.

In Ada 83, the user specified a number D of decimal digits and the implementation then provided model numbers using B binary digits. Intuitively one might expect to need log 10 / log 2 (3.3219...) binary digits for every decimal digit (with appropriate rounding up). The 1980 edition of the Ada Reference Manual (3.5.7 third paragraph) says

(B is the next integer above D*ln(10) / ln(2)).

So 1 decimal digit might be expected to be equivalent to 4 binary digits, 2 decimal digits equivalent to 7 binary digits and so on. But this is not enough. Four binary digits give a relative precision of between 1 in 8 and 1 in 16 whereas one decimal digit requests a maximum precision of 1 in 10. Thus there are places where the model numbers for B = 4 are slightly too far apart.

For example the decimal model numbers around 10000 for D = 1 are

    8000 9000 10000 20000

whereas the binary model numbers for B = 4 are

    7680 8192 9216 10240

and 8192 and 9216 are more than 1000 apart.

This surprising behaviour resulted in the addition of one to the formula so that 3.5.7(6) of Ada 83 concludes

    (The number B is the integer next above (D*log(10) / log(2)) + 1.)

In Ada 95 this formula has been generalized to use T'Machine_Radix rather than 2. However, the special case where Machine_Radix is 10 (or indeed a power of 10) has been overlooked since then no anomalous situations can arise and the "guard digit" is not required.

The formula should therefore be adjusted accordingly.

Note the peculiar phenomenon that more digits may be required for a hexadecimal machine than a decimal machine. Thus one decimal digit requires 2 hexadecimal digits.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0005**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **3.5.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0005 When is a Small clause allowed?

Working Reference Number AI95-00054

Question

3.5.9(8) says, "For a type defined by an ordinary_fixed_point_definition (an ordinary fixed point type), the small may be specified by an attribute_definition_clause (see 13.3)".

3.5.10(2) says, "Small may be specified for nonderived fixed point types via an attribute_definition_clause (see 13.3)".

13.3(5) says, "An attribute_designator is allowed in an attribute_definition_clause only if this International Standard explicitly allows it".

What is the intent? May Small be specified for a derived fixed point type? (No.) May it be specified for a decimal type? (No.)

Summary of Response

A Small clause is illegal for a decimal fixed point type. A Small clause is illegal for a derived fixed point type.

Corrigendum Wording

Replace 3.5.10(2):

S'Small S'Small denotes the *small* of the type of S. The value of this attribute is of the type *universal_real*. Small may be specified for nonderived fixed point types via an attribute_definition_clause (see 13.3); the expression of such a clause shall be static.

by:

S'Small S'Small denotes the *small* of the type of S. The value of this attribute is of the type *universal_real*. Small may be specified for nonderived ordinary fixed point types via an attribute_definition_clause (see 13.3); the expression of such a clause shall be static.

Response

A Small clause is illegal for a decimal fixed point type. A Small clause is illegal for a derived fixed point type.

Discussion

The intent was to disallow a Small clause for a decimal type, because the Small is determined by the type declaration.

The intent was to also disallow a Small clause for a derived fixed point type, because otherwise the model numbers of the parent and derived types might differ, resulting in semantic difficulties.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0006**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **3.6.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0006 The word "prefix" should be in sans serif font

Working Reference Number AI95-00030

Question

Shouldn't the word "prefix" be in the sans serif font? (Yes.)

Summary of Response

The word "prefix" should be in the sans serif font.

Corrigendum Wording

Replace 3.6.2(2):

> The following attributes are defined for a prefix A that is of an array type (after any implicit dereference), or denotes a constrained array subtype:

by:

> The following attributes are defined for a prefix A that is of an array type (after any implicit dereference), or denotes a constrained array subtype:

Discussion

This was an editing error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0007**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.7**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0007 unknown_discriminant_parts on generic formal types

Working Reference Number AI95-00098

Question

12.5(10) (a NOTE) says that "A discriminant_part is allowed only for certain kinds of types, and therefore only for certain kinds of generic formal types. See 3.7."

Unfortunately, the rule in 3.7(8) only applies to known_discriminant_parts. 3.7 does not contain any rule restricting the usage of unknown_discriminant_parts.

Various syntax rules usually do the job, but for generic formal types, the syntax allows unknown_discriminant_parts. Therefore, are the following legal? (No.)

```
generic
    type Disc (<>) is (<>); -- Illegal!
    type Flt  (<>) is digits (<>); -- Illegal!
    type Str  (<>) is new String; -- Illegal!
procedure ....
```

Summary of Response

A generic formal type must not have an unknown_discriminant_part, unless the type is a composite non-array type.

Corrigendum Wording

Replace 3.7(8):

> A known_discriminant_part is only permitted in a declaration for a composite type that is not an array type (this includes generic formal types); a type declared with a known_discriminant_part is called a *discriminated* type, as is a type that inherits (known) discriminants.

by:

> A discriminant_part is only permitted in a declaration for a composite type that is not an array type (this includes generic formal types). A type declared with a known_discriminant_part is called a *discriminated* type, as is a type that inherits (known) discriminants.

Discussion

The intent is that elementary and array types cannot have discriminant parts (known or unknown).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0008**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.7.1; 4.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0008 Aliased objects cannot have discriminants modified

Working Reference Number AI95-00168

Question

Consider the following code fragment:

```
package P is
   pragma Elaborate_Body;
   type T is private;
   A : constant T;
private
   type T (D : Integer := 0) is null record;
   type Ptr is access all T;
   A : constant T := (D => 1);
end P;

with P;
package Q is
   type A1 is array (1 .. 10) of aliased P.T;
   type A2 is array (1 .. 10) of P.T;
   X : A1;
end Q;

with P, Q;
procedure R is
   procedure S (Y : in out Q.A2) is
   begin
      Y (1) := P.A;
   end;
begin
   S (Q.A2 (Q.X)); -- This call will change the discriminant of Q.X (1)
end;
```

This example illustrates a case where it is possible to change the discriminant of an aliased component of an object, which is supposed to be forbidden.

Summary of Response

A view conversion of an array object is illegal if the target subtype and the operand do not have both aliased components or both non-aliased components.

A discriminant constraint for a general access type is illegal if there are places where the designated subtype appears constrained and others where it appears unconstrained.

Corrigendum Wording

Replace 3.7.1(7):

> A discriminant_constraint is only allowed in a subtype_indication whose subtype_mark denotes either an unconstrained discriminated subtype, or an unconstrained access subtype whose designated subtype is an unconstrained discriminated subtype.

by:

> A discriminant_constraint is only allowed in a subtype_indication whose subtype_mark denotes either an unconstrained discriminated subtype, or an unconstrained access subtype whose designated subtype is an unconstrained discriminated subtype. However, in the case of a general access subtype, a discriminant_constraint is illegal if there is a place within the immediate scope of the designated subtype where the designated subtype's view is constrained.

Replace 4.6(11):

- Corresponding index types shall be convertible; and

by:

- Corresponding index types shall be convertible;

Replace 4.6(12):

- The component subtypes shall statically match.

by:

- The component subtypes shall statically match; and
- In a view conversion, the target type and the operand type shall both or neither have aliased components.

Discussion

The problem (1) comes from the fact that it is possible to use a view conversion to convert an array object with aliased components to an array type with non-aliased components. Such a conversion must be disallowed.

The ARG also discussed the following example, which illustrates another case where the standard seems to allow a discriminant to be changed:

```
with Q;
package body P is
   PT : Ptr (0) := Q.X (1)'access;
begin
   Q.X := (others => (D => 2)); -- Changes the discriminant of Q.X (2)
end P;
```

The root of problem (2) is that there are places (e.g., the visible part of P) where P.T is constrained, but other places (e.g., the private part and body of P) where P.T is unconstrained. This causes privacy problems when applying the following rule:

"if a component_definition contains the reserved word aliased and the type of the component is discriminated, then the nominal subtype of the component shall be constrained." (3.6(11))

Also note that the problem exists with non-private types, provided that the characteristic that the type is unconstrained is not visible everywhere:

```
package P.C is
   type NT is new T;
private
   type Ptr is access all NT; -- Causes the same problems as P.Ptr.
end P.C;
```

One way to fix this problem would be to require a component-by-component check on the assignment to Q.X, but that would be very expensive. Moreover, a compile-time check would clearly be better than a run-time check.

Aliasedness of the components is not really what is causing trouble, though. It is really the existence of a general access type, and in fact of a discriminant constraint on such an access type, which causes trouble. Thus, forbidding such a constraint is the chosen solution, especially considering that constraints on access types are not a terribly useful feature.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0009**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.8; 3.11; 9.1; 9.4; 13; 13.1; 13.3; 13.4; 13.11; 13.13.2; 13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0009 Attribute definition clause for stream attributes

Working Reference Number AI95-00137

Question

13.1(10) seems to forbid the following example:

```
with Ada.Streams; use Ada.Streams;
generic
  type T is private;
package Attr_Rep is
  type NT is new T;
  procedure Attribute_Write(
            Stream : access Root_Stream_Type'Class;
            Item   : in NT);
  for NT'Write use Attribute_Write; -- Illegal?  (No.)
end Attr_Rep;
```

Summary of Response

13.1(10) says:

> For an untagged derived type, no type-related representation items are allowed if the parent type is a by-reference type, or has any user-defined primitive subprograms.

This rule does not apply to an attribute_definition_clause for one of the stream-oriented attributes Read, Write, Input, and Output.

Corrigendum Wording

Replace 3.8(5):

component_item ::= component_declaration | representation_clause

by:

component_item ::= component_declaration | aspect_clause

Replace 3.11(4):

basic_declarative_item ::=
  basic_declaration | representation_clause | use_clause

by:

basic_declarative_item ::=
  basic_declaration | aspect_clause | use_clause

Replace 9.1(5):

task_item ::= entry_declaration | representation_clause

by:

task_item ::= entry_declaration | aspect_clause

Replace 9.1(12):

As part of the initialization of a task object, any representation_clauses and any per-object constraints associated with entry_declarations of the corresponding task_definition are elaborated in the given order.

by:

As part of the initialization of a task object, any aspect_clauses and any per-object constraints associated with entry_declarations of the corresponding task_definition are elaborated in the given order.

Replace 9.4(5):

protected_operation_declaration ::= subprogram_declaration
| entry_declaration
| representation_clause

by:

protected_operation_declaration ::= subprogram_declaration
| entry_declaration
| aspect_clause

Replace 9.4(8):

protected_operation_item ::= subprogram_declaration
| subprogram_body
| entry_body
| representation_clause

by:

protected_operation_item ::= subprogram_declaration
| subprogram_body
| entry_body
| aspect_clause

Replace 13(1):

This section describes features for querying and controlling aspects of representation and for interfacing to hardware.

by:

This section describes features for querying and controlling certain aspects of entities and for interfacing to hardware.

Replace the title of 13.1:

Representation Items

by:

Operational and Representation Items

Replace 13.1(1):

There are three kinds of *representation items*: representation_clauses, component_clauses, and *representation pragmas*. Representation items specify how the types and other entities of the language are to be mapped onto the underlying machine. They can be provided to give more efficient representation or to interface with features that are outside the domain of the language (for example, peripheral hardware). Representation items also specify other specifiable properties of entities. A representation item applies to an entity identified by a local_name, which denotes an entity declared local to the current declarative region, or a library unit declared immediately preceding a representation pragma in a compilation.

by:

Representation and operational items can be used to specify aspects of entities. Two kinds of aspects of entities can be specified: aspects of representation and operational aspects. Representation items specify how the types and other entities of the language are to be mapped onto the underlying machine. Operational items specify other properties of entities.

There are six kinds of *representation items*: attribute_definition_clauses for representation attributes, enumeration_representation_clauses, record_representation_clauses, at_clauses, component_clauses, and *representation pragmas*. They can be provided to give more efficient representation or to interface with features that are outside the domain of the language (for example, peripheral hardware).

An *operational item* is an attribute_definition_clause for an operational attribute.

An operational item or a representation item applies to an entity identified by a local_name, which denotes an entity declared local to the current declarative region, or a library unit declared immediately preceding a representation pragma in a compilation.

Replace 13.1(2):

representation_clause ::= attribute_definition_clause
| enumeration_representation_clause
| record_representation_clause
| at_clause

by:

aspect_clause ::= attribute_definition_clause
| enumeration_representation_clause
| record_representation_clause
| at_clause

Replace 13.1(4):

A representation pragma is allowed only at places where a representation_clause or compilation_unit is allowed.

by:

A representation pragma is allowed only at places where an aspect_clause or compilation_unit is allowed.

Replace 13.1(5):

In a representation item, if the local_name is a direct_name, then it shall resolve to denote a declaration (or, in the case of a pragma, one or more declarations) that occurs immediately within the same declarative_region as the representation item. If the local_name has an attribute_designator, then it shall resolve to denote an implementation-defined component (see 13.5.1) or a class-wide type implicitly declared immediately within the same declarative_region as the representation item. A local_name that is a *library_unit*_name (only permitted in a representation pragma) shall resolve to denote the library_item that immediately precedes (except for other pragmas) the representation pragma.

by:

In an operational item or representation item, if the local_name is a direct_name, then it shall resolve to denote a declaration (or, in the case of a pragma, one or more declarations) that occurs immediately within the same declarative_region as the item. If the local_name has an attribute_designator, then it shall resolve to denote an implementation-defined component (see 13.5.1) or a class-wide type implicitly declared immediately within the same declarative_region as the item. A local_name that is a *library_unit*_name (only permitted in a representation pragma) shall resolve to denote the library_item that immediately precedes (except for other pragmas) the representation pragma.

Replace 13.1(6):

The local_name of a representation_clause or representation pragma shall statically denote an entity (or, in the case of a pragma, one or more entities) declared immediately preceding it in a compilation, or within the same declarative_part, package_specification, task_definition, protected_definition, or record_definition as the representation item. If a local_name denotes a local callable entity, it may do so through a local subprogram_renaming_declaration (as a way to resolve ambiguity in the presence of overloading); otherwise, the local_name shall not denote a renaming_declaration.

by:

The local_name of an aspect_clause or representation pragma shall statically denote an entity (or, in the case of a pragma, one or more entities) declared immediately preceding it in a compilation, or within the same declarative_part, package_specification, task_definition, protected_definition, or record_definition as the representation or operational item. If a local_name denotes a local callable entity, it may do so through a local subprogram_renaming_declaration (as a way to resolve ambiguity in the presence of overloading); otherwise, the local_name shall not denote a renaming_declaration.

Insert after 13.1(8):

A representation item *directly specifies* an *aspect of representation* of the entity denoted by the local_name, except in the case of a type-related representation item, whose local_name shall denote a first subtype, and which directly specifies an aspect of the subtype's type. A representation item that names a subtype is either *subtype-specific* (Size and Alignment clauses) or *type-related* (all others). Subtype-specific aspects may differ for different subtypes of the same type.

the new paragraph:

An operational item *directly specifies* an *operational aspect* of the type of the subtype denoted by the local_name. The local_name of an operational item shall denote a first subtype. An operational item that names a subtype is type-related.

Insert after 13.1(9):

A representation item that directly specifies an aspect of a subtype or type shall appear after the type is completely defined (see 3.11.1), and before the subtype or type is frozen (see 13.14). If a representation item is given that directly specifies an aspect of an entity, then it is illegal to give another representation item that directly specifies the same aspect of the entity.

the new paragraph:

An operational item that directly specifies an aspect of a type shall appear before the type is frozen (see 13.14). If an operational item is given that directly specifies an aspect of a type, then it is illegal to give another operational item that directly specifies the same aspect of the type.

Replace 13.1(11):

Representation aspects of a generic formal parameter are the same as those of the actual. A type-related representation item is not allowed for a descendant of a generic formal untagged type.

by:

Operational and representation aspects of a generic formal parameter are the same as those of the actual. A type-related representation item is not allowed for a descendant of a generic formal untagged type.

Replace 13.1(13):

A representation item that is not supported by the implementation is illegal, or raises an exception at run time.

by:

A representation or operational item that is not supported by the implementation is illegal, or raises an exception at run time.

Replace 13.1(19):

For the elaboration of a representation_clause, any evaluable constructs within it are evaluated.

by:

For the elaboration of an aspect_clause, any evaluable constructs within it are evaluated.

Replace the title of 13.3:

Representation Attributes

by:

Operational and Representation Attributes

Replace 13.3(1):

The values of certain implementation-dependent characteristics can be obtained by interrogating appropriate representation attributes. Some of these attributes are specifiable via an attribute_definition_clause.

by:

The values of certain implementation-dependent characteristics can be obtained by interrogating appropriate operational or representation attributes. Some of these attributes are specifiable via an attribute_definition_clause.

Replace 13.3(5):

An attribute_designator is allowed in an attribute_definition_clause only if this International Standard explicitly allows it, or for an implementation-defined attribute if the implementation allows it. Each specifiable attribute constitutes an aspect of representation.

by:

An attribute_designator is allowed in an attribute_definition_clause only if this International Standard explicitly allows it, or for an implementation-defined attribute if the implementation allows it. Each specifiable attribute constitutes an operational aspect or an aspect of representation.

Replace 13.3(9):

The following attributes are defined:

by:

The following representation attributes are defined: Address, Alignment, Size, Storage_Size, and Component_Size.

Replace 13.3(74):

For every subtype S of a tagged type T (specific or class-wide), the following attribute is defined:

by:

The following operational attribute is defined: External_Tag.

For every subtype S of a tagged type T (specific or class-wide):

Replace 13.4(11):

NOTES

11 Unchecked_Conversion may be used to query the internal codes used for an enumeration type. The attributes of the type, such as Succ, Pred, and Pos, are unaffected by the representation_clause. For example, Pos always returns the position number, *not* the internal integer code that might have been specified in a representation_clause.

by:

NOTES

11 Unchecked_Conversion may be used to query the internal codes used for an enumeration type. The attributes of the type, such as Succ, Pred, and Pos, are unaffected by the enumeration_representation_clause. For example, Pos always returns the position number, *not* the internal integer code that might have been specified in an enumeration_representation_clause.

Replace 13.11(12):

For every access subtype S, the following attributes are defined:

by:

For every access subtype S, the following representation attributes are defined:

Replace 13.13.2(1):

The Write, Read, Output, and Input attributes convert values to a stream of elements and reconstruct values from a stream.

by:

The operational attributes Write, Read, Output, and Input convert values to a stream of elements and reconstruct values from a stream.

Replace 13.14(19):

> A representation item that directly specifies an aspect of an entity shall appear before the entity is frozen (see 13.1).

by:

> An operational item or a representation item that directly specifies an aspect of an entity shall appear before the entity is frozen (see 13.1).

Discussion

The intent of 13.1(10) is to forbid two types from having different representation in certain cases. However, the stream-oriented attributes, although they are formally defined to be "representation attributes", do not actually affect the representation of the type. Therefore, there is no need for 13.1(10) to apply to these attributes. Furthermore, as the example illustrates, applying the rule to these attributes would seriously hinder their usefulness.

The definition of stream attributes as "representation attributes" has proven to be a continuing problem. Several issues have made it necessary to exempt stream attributes from the rules for representation attributes; indeed the number of such exemptions makes it clear that it is confusing to classify them as representation attributes. Therefore, we have taken the major step of defining a new kind of attribute, the "operational attributes", and redefining stream attributes to be of this kind.

In particular, 7.3(5), 13.1(10), and the last sentence of 13.1(11) are unchanged, so that these rules do not apply to operational items. None of these rules are necessary for these attributes. We've also left 3.8(11) unchanged, as an operational item cannot occur here. Changes to 13.1(15) and 13.1(18) are found in 8652/0040 (AI-00108).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0010**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.9.2; 3.10.2; 4.8**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0010 Expected type of a 'Access attribute

Working Reference Number AI95-00127

Question

Consider the following code fragment:

```
type T is tagged null record;
procedure P(X : access T);
Y : aliased T'Class := ...;
type T_Ptr is access all T'Class;
Z : T_Ptr;

P(Y'access); -- (1) Legal?  (Yes.)
P(new T'Class'(...)); -- (2) Legal?  (Yes.)
P(Z); -- (3) Legal.
```

The call at (3) is clearly legal, and is a dispatching call.

However, the call at (1) appears to be illegal. The expected type for Y'Access is the anonymous type "access T", by 6.4.1(3). 3.10.2(24) says, "If the designated type of A [here, A is the anonymous access type] is tagged, then the type of the view [Y] shall be covered by the designated type". The type of the view is T'Class, which is not covered by the designated type, which is T. Therefore, Y'Access is illegal here.

The call at (2) appears to be illegal for the same reason.

It would seem that the same rules should apply to all of these calls; (1) and (2) should be legal, and should be dispatching calls.

Summary of Response

An attribute reference of the Access attribute may be used as the actual parameter in a dispatching call, if the formal is an access parameter designating a tagged type, and the prefix of the attribute reference is of the corresponding class-wide type. Such an actual is considered to be dynamically tagged.

An analogous rule applies to an attribute reference of Unchecked_Access and to an allocator.

Corrigendum Wording

Replace 3.9.2(7):

> A type_conversion is statically or dynamically tagged according to whether the type determined by the subtype_mark is specific or class-wide, respectively. For a controlling operand that is designated by an actual parameter, the controlling operand is statically or dynamically tagged according to whether the designated type of the actual parameter is specific or class-wide, respectively.

by:

> A type_conversion is statically or dynamically tagged according to whether the type determined by the subtype_mark is specific or class-wide, respectively. For an object that is designated by an expression whose expected type is an anonymous access-to-specific tagged type, the object is dynamically tagged if the expression, ignoring enclosing parentheses, is of the form X'Access, where X is of a class-wide type, or is of the form **new** T'(...), where T denotes a class-wide subtype. Otherwise, the object is statically or dynamically tagged according to whether the designated type of the type of the expression is specific or class-wide, respectively.

Replace 3.9.2(9):

> If the expected type for an expression or name is some specific tagged type, then the expression or name shall not be dynamically tagged unless it is a controlling operand in a call on a dispatching operation. Similarly, if the expected type for an expression is an anonymous access-to-specific tagged

type, then the expression shall not be of an access-to-class-wide type unless it designates a controlling operand in a call on a dispatching operation.

by:

If the expected type for an expression or name is some specific tagged type, then the expression or name shall not be dynamically tagged unless it is a controlling operand in a call on a dispatching operation. Similarly, if the expected type for an expression is an anonymous access-to-specific tagged type, then the object designated by the expression shall not be dynamically tagged unless it is a controlling operand in a call on a dispatching operation.

Replace 3.10.2(24):

X'Access

X'Access yields an access value that designates the object denoted by X. The type of X'Access is an access-to-object type, as determined by the expected type. The expected type shall be a general access type. X shall denote an aliased view of an object, including possibly the current instance (see 8.6) of a limited type within its definition, or a formal parameter or generic formal object of a tagged type. The view denoted by the prefix X shall satisfy the following additional requirements, presuming the expected type for X'Access is the general access type *A*:

by:

X'Access

X'Access yields an access value that designates the object denoted by X. The type of X'Access is an access-to-object type, as determined by the expected type. The expected type shall be a general access type. X shall denote an aliased view of an object, including possibly the current instance (see 8.6) of a limited type within its definition, or a formal parameter or generic formal object of a tagged type. The view denoted by the prefix X shall satisfy the following additional requirements, presuming the expected type for X'Access is the general access type *A*, with designated type *D*:

Replace 3.10.2(27):

- If the designated type of *A* is tagged, then the type of the view shall be covered by the designated type; if *A*'s designated type is not tagged, then the type of the view shall be the same, and either *A*'s designated subtype shall statically match the nominal subtype of the view, or the designated subtype shall be discriminated and unconstrained;

by:

- If *A* is a named access type and *D* is a tagged type, then the type of the view shall be covered by *D*; if *A* is anonymous and *D* is tagged, then the type of the view shall be either *D*'Class or a type covered by *D*; if *D* is untagged, then the type of the view shall be *D*, and *A*'s designated subtype shall either statically match the nominal subtype of the view or be discriminated and unconstrained;

Replace 4.8(3):

The expected type for an allocator shall be a single access-to-object type whose designated type covers the type determined by the subtype_mark of the subtype_indication or qualified_expression.

by:

The expected type for an allocator shall be a single access-to-object type with designated type *D* such that either *D* covers the type determined by the subtype_mark of the subtype_indication or qualified_expression, or the expected type is anonymous and the determined type is *D*'Class.

Discussion

The rules should be equivalent in these cases; anything else would be surprising to the programmer. This is achieved by the above wording.

In the call at (1), Y'Access is of the anonymous type "access T". Y'Access is dynamically tagged, despite the fact that its type's designated type is not class-wide.

In the call at (2), new T'Class'(...) is also of the anonymous type "access T", and is also dynamically tagged.

Thus, all three calls are legal, and are dispatching calls to P.

No wording changes are needed for Unchecked_Access, since it is already defined in terms of Access.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0011**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.9.2; 6.3.1; 13.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0011 Calling conventions

Working Reference Number AI95-00117

Question

6.3.1(2-13) define the default convention of various entities (that is, the convention in the absence of a convention-specifying pragma):

> As explained in B.1, ‘‘Interfacing Pragmas’’, a convention can be specified for an entity. For a callable entity or access-to-subprogram type, the convention is called the calling convention. The following conventions are defined by the language:
>
> > The default calling convention for any subprogram not listed below is Ada. A pragma Convention, Import, or Export may be used to override the default calling convention (see B.1).
> >
> > The Intrinsic calling convention represents subprograms that are ‘‘built in’’ to the compiler. The default calling convention is Intrinsic for the following:
> >
> > an enumeration literal;
> >
> > a "/=" operator declared implicitly due to the declaration of "=" (see 6.6);
> >
> > any other implicitly declared subprogram unless it is a dispatching operation of a tagged type;
> >
> > an inherited subprogram of a generic formal tagged type with unknown discriminants;
> >
> > an attribute that is a subprogram;
> >
> > a subprogram declared immediately within a protected_body.
> >
> > The Access attribute is not allowed for Intrinsic subprograms.
>
> The default calling convention is protected for a protected subprogram, and for an access-to-subprogram type with the reserved word protected in its definition.
>
> The default calling convention is entry for an entry.

---

1. What is the default convention of an entity not covered by 6.3.1, such as a record type? (Ada.)

---

2. Does an inherited or overriding subprogram have (by default) the same convention as the parent subprogram? (Yes.)

6.3.1(3) implies that if the calling convention of a parent subprogram is not Ada, the default convention of an overriding subprogram is, nonetheless, Ada. However, 3.9.2(10) says:

> If the dispatching operation overrides an inherited subprogram, it shall be subtype conformant with the inherited subprogram.

6.3.1(17) requires matching conventions for subtype conformance. Thus, the *default* calling convention for this overriding case is illegal; the programmer *must* give a pragma specifying the convention in this case. This seems unfriendly.

On the other hand, 3.4(18) says:

> The profile of an inherited subprogram (including an inherited enumeration literal) is obtained from the profile of the corresponding (user-defined) primitive subprogram of the parent type, after systematic replacement of each subtype of its profile (see 6.1) that is of the parent type with a corresponding subtype of the derived type. ...

And 6.1(22) says:

> Associated with a profile is a calling convention.

These paragraphs seem to imply that an inherited subprogram inherits the calling convention of its parent, as part of the inherited profile.

---

3. Is an implicitly declared dispatching "/=" operator legal? (Yes.)

Paragraph 3.9.1(1) says that the primitive subprograms of a tagged type are called dispatching operations. Paragraph 3.9.2(10) goes on to say that a dispatching operation shall not be of convention Intrinsic. However, paragraph 6.3.1(6) says that "/=" declared implicitly due to the declaration of "=" is of convention Intrinsic, by default.

Together these imply that the "/=" implicitly declared due to the declaration of "=" of a tagged type is an illegal dispatching operation. Is this the intent? (No.)

Summary of Response

Unless specified otherwise in the standard, the default convention of any entity is Ada.

An inherited or overriding subprogram of a type extension inherits the calling convention of the parent subprogram.

New operations of type extensions have the convention of their type unless a new convention is defined for the operation, if this is supported by an implementation.

The convention of the partial view of a private type or private extension is the convention of the full type.

An explicitly declared dispatching operation shall not have convention Intrinsic. However, an implicitly declared dispatching "/=" operator with Boolean result legally has convention Intrinsic.

Corrigendum Wording

Replace 3.9.2(10):

> In the declaration of a dispatching operation of a tagged type, everywhere a subtype of the tagged type appears as a subtype of the profile (see 6.1), it shall statically match the first subtype of the tagged type. If the dispatching operation overrides an inherited subprogram, it shall be subtype conformant with the inherited subprogram. A dispatching operation shall not be of convention Intrinsic. If a dispatching operation overrides the predefined equals operator, then it shall be of convention Ada (either explicitly or by default — see 6.3.1).

by:

> In the declaration of a dispatching operation of a tagged type, everywhere a subtype of the tagged type appears as a subtype of the profile (see 6.1), it shall statically match the first subtype of the tagged type. If the dispatching operation overrides an inherited subprogram, it shall be subtype conformant with the inherited subprogram. The convention of an inherited or overriding dispatching operation is the convention of the corresponding primitive operation of the parent type. An explicitly declared dispatching operation shall not be of convention Intrinsic.

Replace 6.3.1(2):

As explained in B.1, ‘‘Interfacing Pragmas’’, a *convention* can be specified for an entity. For a callable entity or access-to-subprogram type, the convention is called the *calling convention*. The following conventions are defined by the language:

by:

As explained in B.1, ‘‘Interfacing Pragmas’’, a *convention* can be specified for an entity. Unless this International Standard states otherwise, the default convention of an entity is Ada. For a callable entity or access-to-subprogram type, the convention is called the *calling convention*. The following calling conventions are defined by the language:

Insert after 6.3.1(13):

- The default calling convention is *entry* for an entry.

the new paragraph:

- If not specified above as Intrinsic, the calling convention for any inherited or overriding dispatching operation of a tagged type is that of the corresponding subprogram of the parent type. The default calling convention for a new dispatching operation of a tagged type is the convention of the type.

Replace 13.1(11):

Representation aspects of a generic formal parameter are the same as those of the actual. A type-related representation item is not allowed for a descendant of a generic formal untagged type.

by:

Representation aspects of a generic formal parameter are the same as those of the actual. Representation aspects of a partial view are the same as those of the full view. A type-related representation item is not allowed for a descendant of a generic formal untagged type.

Discussion

1. The default convention ought to be Ada for any entity not covered by 6.3.1. The dispatching operations of a type ought to inherit the convention of the type, for convenient interfacing to other OOP languages. (See below for more discussion of this point.)

---

2. It is important that Ada allow clean interfaces to other programming languages. In particular, it is important that Ada's tagged types can be used to interface to other OOP languages.

If an Ada implementation is tightly integrated with another language, such as C++ or Java, it is nice if an Ada tagged type can be declared as an extension of a (foreign) type (or class) of the other language. Presumably, all of the dispatching operations of this foreign type would be defined as imported, with the convention of that other language. When defining the type extension in Ada, it would be very inconvenient if every overriding needed a pragma Convention on it to match that of the inherited operation, as required by 3.9.2(10).

Hence, it seems appropriate to define the default calling convention of an overriding of an inherited dispatching operation to be the same as that of the corresponding operation on the parent type, rather than always being convention "Ada" as specified in 6.3.1(3).

For example:

```
package Java.Graphics is
    type Graphics_Obj is tagged limited private;
    procedure drawString(G : in out Graphics_Obj; S : String);
    pragma Import(Java, drawString);
  ...
end Java.Graphics;

with Java.Graphics; use Java.Graphics;
```

```
package Flight_Simulator is
    type Simulator_Obj is new Graphics_Obj with private;
    procedure drawString(S : in out Simulator_Obj; S : String);
    -- implicit: pragma Convention(Java, drawString);
 ...
end Flight_Simulator;
```

The "pragma Convention(Java, ...);" should be implicit when overriding a dispatching operation with convention Java. Anything else would be illegal by 3.9.2(10), and it seems silly to require the programmer to litter their program with redundant "pragma Convention"s.

The Note B.1(42) - derived from 13.1 - implies permission of implementation-defined restrictions of interfacing pragmas. Hence an implementation will be allowed to reject the attempt to create "heterogeneous" tagged types, i.e., types having primitive operations of different, explicitly specified conventions or of explicitly specified conventions different from the convention of the type.

2.a. The "Breach of Privacy" Issue

Presently, the convention of a primitive subprogram can be specified in the private part of the declaring package. The current rules require explicit confirmation of this convention for overriding subprograms and thus constitute a breach of the privacy of the private part, since the user needs to know about this privately specified convention in order to make the overriding declaration legal.

We are very reluctant to mandate Convention pragmas in the visible part of the package. Although such a rule might be derivable from freezing rules, it nevertheless could create a compatibility problem for existing code.

The proposed new rule of inheriting the convention eases, but does not eliminate, the breach of privacy, as any explicitly specified convention will still need to confirm the inherited convention.

At the implementation level, both the existing and the proposed model breach the private part, as subtype conformance of overriding with inherited subprograms includes checking for equality of the convention.

2.b. Deriving the convention of operations from the type

Having dispatching operations with the convention of some other OOP language, while the type is not represented according to the convention of this other language, will be almost impossible to implement. The "normal" case will be that both type and operations need the convention pragma. In this context, it makes little sense that the convention of primitive operations defaults to Ada rather than to the convention of their type. The user will be forced to repeat the pragma for all the operations of the type. Considerably more convenient is a model, in which the default convention of dispatching operations is inherited from the type, but overridable if the implementation allows for such mixed conventions.

Since current rules imply that the convention of a type needs to be specified for the full view of the type, such dependency creates yet another breach of privacy in the case of private tagged types. However, the breach already exists as explained in 2.a. and then to exploit it for more convenience to the user and a cleaner overall model seems justified.

Mandating the specification of the convention for the partial view in order to avoid the breach of privacy seems too much of an incompatibility for existing code.

2.c. The convention of a partial view

As mentioned previously, the current rules of the standard require that the convention be specified on the full view. So, what is the convention of a partial view? It is clear that a partial view and full view must have the same representation (including convention), since they are just views of the same entity. However, while this is obvious, it is also not mentioned in the standard. A statement to this effect needs to be added to 13.1.

---

3. Clearly, an implicitly declared dispatching "/=" should not automatically be illegal.

The proposed new wording precludes declaring a dispatching operation by renaming the Intrinsic "/=", which is good (since there is no real body associated with "/="). It does not make "/=" itself illegal, which is also good.

The reason for 6.3.1(4-10) making various subprograms Intrinsic is that these subprograms don't really exist in machine code. For example, an implementation would typically not generate any code for the implicitly-declared "/=" operator -- instead, it would call the "=" operator, and then do a "not" operation at the call site. We don't want to allow 'Access of such subprograms, because it would introduce an implementation burden -- the implementation would have to materialize these subprograms as real machine-code subprograms, which is not otherwise necessary.

A similar issue arises with 6.3.1, which says that an inherited subprogram of a generic formal type with unknown discriminants is of convention Intrinsic, by default.

The reason for this rule is obscure enough that it should have been documented in the AARM: Consider:

```
package P is
    type Root is tagged null record;
    procedure Proc(X: Root);
end P;

generic
    type Formal(<>) is new Root with private;
package G is
    ...
end G;

package body G is
    ...
    X: Formal := ...;
    ...
    Proc(X); -- This is a dispatching call in Instance, because
             -- the actual type for Formal is class-wide.
    ...
    -- Proc'Access would be illegal here, because it is of
    -- convention Intrinsic, by 6.3.1(8).
end G;

type Actual is new Root with ...;
procedure Proc(X: Actual);
package Instance is new G(Formal => Actual'Class);
    -- It is legal to pass in a class-wide actual, because Formal
    -- has unknown discriminants.
```

Within Instance, all calls to Proc will be dispatching calls, so Proc doesn't really exist in machine code, so we wish to avoid taking 'Access of it. 6.3.1(8) applies to those cases where the actual type might be class-wide, and makes these Intrinsic, thus forbidding 'Access.

The wording change to 3.9.2(10) shown above means that it is permitted to have such an inherited subprogram. If the specification of G contained a type extension of Formal, then that type's inherited Proc would also have convention Intrinsic, which would be legal. However, an explicit overriding of that Proc would be illegal.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0012**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0012 Derived access types share the same pool

Working Reference Number AI95-00062

Question

NOTE 3.4(31) says, "If the parent type is an access type, then the parent and the derived type share the same storage pool..." This is clearly what we want, but I can't seem to prove it from the real rules (i.e. non-NOTES).

Summary of Response

A derived access type shares its parent's storage pool.

Corrigendum Wording

Replace 3.10(7):

> There are two kinds of access types, *access-to-object* types, whose values designate objects, and *access-to-subprogram* types, whose values designate subprograms. Associated with an access-to-object type is a *storage pool*; several access types may share the same storage pool. A storage pool is an area of storage used to hold dynamically allocated objects (called *pool elements*) created by allocators; storage pools are described further in 13.11, ‘‘Storage Management’’.

by:

> There are two kinds of access types, *access-to-object* types, whose values designate objects, and *access-to-subprogram* types, whose values designate subprograms. Associated with an access-to-object type is a *storage pool*; several access types may share the same storage pool. All descendants of an access type share the same storage pool. A storage pool is an area of storage used to hold dynamically allocated objects (called *pool elements*) created by allocators; storage pools are described further in 13.11, ‘‘Storage Management’’.

Response

A derived access type shares its parent's storage pool.

Discussion

NOTE 3.4(31) makes the intent clear.

Furthermore, 13.11.2(16) says, "The execution of a call to an instance of Unchecked_Deallocation is erroneous if the object was created other than by an allocator for an access type whose pool is Name'Storage_Pool."

Thus, if a derived access type does not have the same pool as its parent, then the following would be erroneous:

```
type Parent is access Integer;
type Derived is new Parent;
X: Derived := new Integer;
Y: Parent := Parent(X);
procedure Free is new
    Unchecked_Deallocation(Object => Integer, Name => Parent);
...
Free(Y);
```

The above was not erroneous in Ada 83. This would be a serious upward incompatibility, which there was no intention to introduce.

Note that no such upward incompatibility is documented in the AARM.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0013**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **3.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0013 The first subtype of a type defined by an access[_type]_definition

Working Reference Number AI95-00012

Question

3.10(14) says:

> All subtypes of an access-to-subprogram type are constrained. The first subtype of a type defined by an access_type_definition or an access_to_object_definition is unconstrained if the designated subtype is an unconstrained array or discriminated type; otherwise it is constrained.

However, access_type_definition includes access_to_object_definition. What is the intent?

Summary of Response

The second sentence of 3.10(14) applies to all access-to-object types, including those defined by access_definitions.

Corrigendum Wording

Replace 3.10(14):

> All subtypes of an access-to-subprogram type are constrained. The first subtype of a type defined by an access_type_definition or an access_to_object_definition is unconstrained if the designated subtype is an unconstrained array or discriminated type; otherwise it is constrained.

by:

> All subtypes of an access-to-subprogram type are constrained. The first subtype of a type defined by an access_definition or an access_to_object_definition is unconstrained if the designated subtype is an unconstrained array or discriminated subtype; otherwise it is constrained.

Discussion

The notion of designated subtype doesn't make sense for access-to-subprograms. The intent is that this rule should apply to all access-to-object types. Apparently, access_type_definition is a "typo".

Another typo was noted in this paragraph. The paragraph says "...if the designated subtype is an unconstrained array or discriminated type...", but this clearly should say "...discriminated {sub}type...".

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0014**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **3.11; 3.11.1; 8.5.4; 13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0014 Elaboration checks for renamings-as-body

Working Reference Number AI95-00064

Question

3.11(10) indicates that an elaboration check is required only when a subprogram has an explicit body that is a subprogram_body. However, when a renaming declaration is used as a body, it is possible for the elaboration of the renaming declaration to require the evaluation of a name, such as X.all, that implies some sort of elaboration check should be performed. For example:

```
function F return Integer;
type Pointer_To_Func is access function return Integer;

X : Pointer_To_Func := Q'access;  -- Presume Q already elaborated

Y : Integer := F;  -- We need some sort of elaboration check

....

Z : Pointer_To_Func := X;

function F return Integer renames Z.all;
```

In the above, clearly we need to wait until the expression "Z.all" is evaluated before F can be safely called. However, it is not clear that any check for this is required by 3.11(10).

By the way, where is "body" defined? It presumably includes entry body, and perhaps renaming-as-body. However, only the syntactic entity BODY is defined (in 3.11). Where is the unbolded term "body" defined?

Summary of Response

An elaboration check is performed for a call to a subprogram whose body is given as a renaming-as-body. This check fails if the renaming-as-body has not yet been elaborated. (As usual, an elaboration check is also performed for the renamed subprogram, and fails if *its* body has not yet been elaborated.)

Corrigendum Wording

Replace 3.11(10):

- For a call to a (non-protected) subprogram that has an explicit body, a check is made that the subprogram_body is already elaborated. This check and the evaluations of any actual parameters of the call are done in an arbitrary order.

by:

- For a call to a (non-protected) subprogram that has an explicit body, a check is made that the body is already elaborated. This check and the evaluations of any actual parameters of the call are done in an arbitrary order.

Replace 3.11.1(1):

Declarations sometimes come in two parts. A declaration that requires a second part is said to *require completion*. The second part is called the *completion* of the declaration (and of the entity declared), and is either another declaration, a body, or a pragma.

by:

Declarations sometimes come in two parts. A declaration that requires a second part is said to *require completion*. The second part is called the *completion* of the declaration (and of the entity declared), and is either another declaration, a body, or a pragma. A *body* is a body, an entry_body, or a renaming-as-body (see 8.5.4).

Insert before 8.5.4(8):

For a call on a renaming of a dispatching subprogram that is overridden, if the overriding occurred before the renaming, then the body executed is that of the overriding declaration, even if the overriding declaration is not visible at the place of the renaming; otherwise, the inherited or predefined subprogram is called.

the new paragraph:

For a call to a subprogram whose body is given as a renaming-as-body, the execution of the renaming-as-body is equivalent to the execution of a subprogram_body that simply calls the renamed subprogram with its formal parameters as the actual parameters and, if it is a function, returns the value of the call.

Replace 13.14(3):

The end of a declarative_part, protected_body, or a declaration of a library package or generic library package, causes *freezing* of each entity declared within it, except for incomplete types. A noninstance body causes freezing of each entity declared before it within the same declarative_part.

by:

The end of a declarative_part, protected_body, or a declaration of a library package or generic library package, causes *freezing* of each entity declared within it, except for incomplete types. A noninstance body other than a renaming-as-body causes freezing of each entity declared before it within the same declarative_part.

Discussion

Since the elaboration of a renaming-as-body may evaluate expressions, it is clearly necessary that this elaboration be performed before calling the subprogram. Therefore, an elaboration check should be done on a subprogram whose body is a renaming-as-body, not just when the body is a subprogram_body.

It seems that the right model for renaming-as-body that occurs after the subprogram is frozen should be that of a wrapper subprogram, with its own elaboration flag.

Taken together, these rules imply that when calling a subprogram whose body is a renaming-as-body, a check will be made that the renaming-as-body has been elaborated, and also that the body of the renamed subprogram has been elaborated. Furthermore, if the renamed subprogram is in turn completed by *another* renaming-as-body, the body of that third subprogram will also be checked; the rule is transitive.

See 8652/0027 (AI-00135) for a discussion of circularities involving renamings-as-body.

This issue also adds the missing definition of the semantic term "body". This change makes a renaming-as-body a body. However, doing so triggers the freezing rule 13.14(3): "A noninstance body other than a renaming-as-body causes freezing of each entity declared before it within the same declarative_part." It clearly was the intent of the designers of the language that renaming-as-body not freeze (otherwise the second sentence of 8.5.4(5) could never be true), and existing compilers do not freeze when a renaming-as-body is encountered. We do not want to change this behavior, so we add an exception to 13.14(3).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0015**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **4.1.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0015 Float_Type'Small

Working Reference Number AI95-00093

Question

Paragraph 4.1.4(12) says:

> An implementation may provide implementation-defined attributes; the identifier for an implementation-defined attribute shall differ from those of the language-defined attributes.

AARM J(1.d) lists several Ada 83 floating-point attributes that have been removed from the language, including 'Small; AARM J(1.h), however, says that "Implementations can continue to support the above features for upward compatibility".

Since 'Small is a language-defined attribute for fixed-point types, 4.1.4(12) implies that an implementation must not provide a 'Small attribute for floating-point types. This clearly contradicts the intent of J(1).

May an implementation support the 'Small attribute for floating-point types? (Yes.)

Summary of Response

An implementation may support an implementation-defined attribute Small for floating point types.

Corrigendum Wording

Replace 4.1.4(12):

> An implementation may provide implementation-defined attributes; the identifier for an implementation-defined attribute shall differ from those of the language-defined attributes.

by:

> An implementation may provide implementation-defined attributes; the identifier for an implementation-defined attribute shall differ from those of the language-defined attributes unless supplied for compatibility with a previous edition of this International Standard.

Discussion

The intent is that implementations be allowed to support all Ada 83 attributes, for upward compatibility. Thus, it is important that they be allowed to support the Small attribute on floating point types. Therefore, this resolution makes a specific exception to the rule in 4.1.4(12).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0016**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **4.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0016 Equality for composite types

Working Reference Number AI95-00123

Question

The following language-defined types are private, and have an explicitly defined primitive "=" operator:

> System.Address
>
> Ada.Strings.Maps.Character_Set
>
> Ada.Strings.Bounded.Generic_Bounded_Length.Bounded_String
>
> Ada.Strings.Unbounded.Unbounded_String
>
> Ada.Strings.Wide_Maps.Wide_Character_Set
>
> Ada.Task_Identification.Task_ID

This would seem to imply that the composability of these "=" operators depends on whether the implementation chooses to implement them as tagged types, by 4.5.2(14-15):

> For a type extension, predefined equality is defined in terms of the primitive (possibly user-defined) equals operator of the parent type and of any tagged components of the extension part, and predefined equality for any other components not inherited from the parent type.
>
> For a private type, if its full type is tagged, predefined equality is defined in terms of the primitive equals operator of the full type; if the full type is untagged, predefined equality for the private type is that of its full type.

and by 4.5.2(21-24):

> Given the above definition of matching components, the result of the predefined equals operator for composite types (other than for those composite types covered earlier) is defined as follows:
>
> > If there are no components, the result is defined to be True;
> >
> > If there are unmatched components, the result is defined to be False;
> >
> > Otherwise, the result is defined in terms of the primitive equals operator for any matching tagged components, and the predefined equals for any matching untagged components.

This would cause portability problems.

Also, in the above definition, what does "in terms of" mean? For a composite type, if some parts have an "=" with side effects, does the language define whether all of these side effects happen, and in what order?

Summary of Response

The primitive equality operators of a language-defined type compose properly (i.e., do not "reemerge"), when the type is used as a component type, or a generic actual type.

For any composite type, the order in which "=" is called for components is not defined by the language. Furthermore, if the result can be determined before calling "=" on some components, the language does not define whether "=" is called on those components.

Corrigendum Wording

Insert after 4.5.2(24):

- Otherwise, the result is defined in terms of the primitive equals operator for any matching tagged components, and the predefined equals for any matching untagged components.

the new paragraph:

For any composite type, the order in which "=" is called for components is unspecified. Furthermore, if the result can be determined before calling "=" on some components, it is unspecified whether "=" is called on those components.

Insert after 4.5.2(32):

A membership test using **not in** gives the complementary result to the corresponding membership test using **in**.

the new paragraph:

Implementation Requirements

For all nonlimited types declared in language-defined packages, the "=" and "/=" operators of the type shall behave as if they were the predefined equality operators for the purposes of the equality of composite types and generic formal types.

Discussion

Composability of equality for a type T means three things:

1. If a composite type has a component of type T with a user-defined equality operator, then the predefined equality of the composite type calls the user-defined equality operator of type T (for that component).

2. If an actual type T for a generic formal type has a user-defined equality operator, then the predefined equality on the generic formal type calls the user-defined equality operator of type T.

3. If a parent type T has a user-defined equality operator, then the predefined equality of a type extension of T calls the user-defined equality on T (for the parent part), in addition to comparing the extension parts.

Non-composability means that the predefined equality is called for T, despite the fact that T has a user-defined equality operator. Of course, if there is no user-defined equality, then equality always composes properly.

Item 3 is irrelevant here, since none of the types in question is (visibly) tagged.

For a private type, if the underlying type is tagged, or if there is no user-defined equality, then equality composes. Otherwise, it does not. (Here, "underlying type" means the full type, or if that comes from a private type, then the underlying type of *that* type, and so on.)

However, for the private types mentioned in the question, the standard does not specify whether the underlying type is tagged, nor whether the equality operator is truly user-defined (as opposed to just being the normal bit-wise equality).

It is important that the composability of "=" for these types be defined by the language. We choose to make them composable. An implementation can achieve this by making the full type tagged. Alternatively, the implementation could simply use the predefined "=" for these types. (Alternatively, an implementation could treat these types specially, making them untagged, but with composable equality. However, this would add some complexity to the compiler.)

Here is an analysis of implementation concerns for each type in question:

- System.Address: The intent is for this type to directly represent a hardware address. Therefore, it is probably not feasible to implement it as a tagged type. The simplest implementation of equality of Addresses is thus the normal bit-wise equality. This is what most users would expect, anyway.

    On certain segmented architectures, it is possible for two different addresses to point to the same location. The same thing can happen due to memory mapping, on many machines. Such addresses will typically compare unequal, despite the fact that they point to the same location.

- Ada.Strings.Maps.Character_Set: A typical implementation will use an array of Booleans, so bit-wise equality will be used, so it will compose.

- Ada.Strings.Bounded.Generic_Bounded_Length.Bounded_String: Two reasonable implementations are: (1) Set the unused characters to some particular character, and use bit-wise equality, and (2) use a tagged type with a user-defined equality. Either way, equality will compose. This is, admittedly, a slight implementation burden, because it rules out an untagged record with user-defined equality.

- Ada.Strings.Unbounded.Unbounded_String: A tagged (controlled) type will normally be necessary anyway, for storage reclamation. In a garbage-collected implementation, a tagged type is not strictly necessary, but we choose to require composability anyway.

- Ada.Strings.Wide_Maps.Wide_Character_Set: Some sort of data structure built out of access types is necessary anyway, so the extra overhead of composability is not a serious problem; the implementation can simply make the full type tagged.

- Ada.Task_Identification.Task_ID: This will typically be a pointer-to-TCB of some sort (access-to-TCB, or index-into-table-of-TCB's). In any case, bit-wise equality will work, so equality will compose.

As to the second question, the standard clearly does not define any order of calling "=" on components, nor does it say whether the results are combined with "and" or "and then". Equality operators with side effects are questionable in any case, so we allow implementations freedom to do what is most convenient and/or most efficient. Consider equality of a variant record: The implementation might first check that the discriminants are equal, and if not, skip the component-by-component comparison. Alternatively, the implementation might first compare the common elements, and *then* check the discriminants. A third possibility is to first compare some portions with a bit-wise equality, and then (conditionally) call user-defined equality operators on the other components. All of these implementations are valid.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0017**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **4.6; 8.5.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0017 Definiteness and type derivation

Working Reference Number AI95-00184

Question

The definiteness of a type is not preserved by type derivation. A type with defaulted discriminants may be derived from a type without defaulted discriminants and vice-versa.

This makes it possible to rename a component of a record that later disappears due to an assignment to the enclosing object, as shown in the following examples:

1 - An example where the parent type is indefinite and the derived type is definite:

```
type T1 (D1 : Boolean) is
   record
      case D1 is
         when False =>
            C1 : Integer;
         when True =>
            C2 : Float;
         end case;
      end record;

generic
   type F is new T1;
   X : in out F;
package G is
   C1_Ren : Integer renames X.C1;
end G;

type T2 (D2 : Boolean := True) is new T1 (D1 => D2);

Y : T2;

package I is new G (T2, Y);

Y := (D1 => True, C2 => 3.0); -- Oops!  What happened to I.C1_Ren
```

The declaration of C1_Ren in the generic G is legal as per 8.5.1(5), because the formal type F is indefinite. But when G is instantiated with type T2, the actual type is definite, so now we have renamed a component that may disappear by assignment to the variable Y. Note that the declaration of C1_Ren might be in the body of G, so we cannot avoid this problem by rechecking 8.5.1(5) on the instantiation.

2 - An example where the parent type is definite and the derived type is indefinite:

```
type Definite_Parent (D1 : Integer := 6) is
   record
      F : String (1 .. D1);
   end record;

type Indefinite_Child (D2 : Integer) is new Definite_Parent (D1 => D2);

Y : Definite_Parent;

procedure P (X : in out Indefinite_Child) is
   C : Character renames X.F (3);
begin
   X := (0, "");
   -- X.F (3) has disappeared!
end;
begin
P (Indefinite_Child (Y));
```

Assume that the implementation chooses to pass X by reference. Then, 6.4.1(10) says that there is an implicit view conversion to Indefinite_Child, and the formal parameter X then denotes the result of this view

conversion. The result of the explicit view conversion is unconstrained, and the result of the implicit view conversion is also unconstrained, hence X is unconstrained, which violates the language design principle of the NOTE in 3.7(28).

One of the unpleasant consequences of this violation is that the assignment to X doesn't raise an exception, and after the execution of this assignment C denotes a non-existent component.

Note that if the implementation chooses to pass by copy, then there is an implicit value conversion -- see 6.4.1(11). So in that case, there's no problem.

Summary of Response

The legality rules about object renaming are checked in the private part of an instance. In a generic body, they are checked in an assume-the-worst manner: it is illegal to rename a component that depends on a discriminant of a variable whose nominal subtype is an untagged indefinite generic formal derived type (or a descendant of such a type) unless the variable is aliased.

A view conversion to an indefinite object is constrained.

For a conversion of an object name to a tagged type to be a view conversion, the object's nominal subtype has to be tagged.

Corrigendum Wording

Replace 4.6(5):

A type_conversion whose operand is the name of an object is called a *view conversion* if its target type is tagged, or if it appears as an actual parameter of mode **out** or **in out**; other type_conversions are called *value conversions*.

by:

A type_conversion whose operand is the name of an object is called a *view conversion* if both its target type and operand type are tagged, or if it appears as an actual parameter of mode **out** or **in out**; other type_conversions are called *value conversions*.

Replace 4.6(54):

- If the target type is composite, the bounds or discriminants (if any) of the view are as defined above for a value conversion; each nondiscriminant component of the view denotes the matching component of the operand object; the subtype of the view is constrained if either the target subtype or the operand object is constrained, or if the operand type is a descendant of the target type, and has discriminants that were not inherited from the target type;

by:

- If the target type is composite, the bounds or discriminants (if any) of the view are as defined above for a value conversion; each nondiscriminant component of the view denotes the matching component of the operand object; the subtype of the view is constrained if either the target subtype or the operand object is constrained, or if the target subtype is indefinite, or if the operand type is a descendant of the target type and has discriminants that were not inherited from the target type;

Replace 8.5.1(5):

The renamed entity shall not be a subcomponent that depends on discriminants of a variable whose nominal subtype is unconstrained, unless this subtype is indefinite, or the variable is aliased. A slice of an array shall not be renamed if this restriction disallows renaming of the array.

by:

The renamed entity shall not be a subcomponent that depends on discriminants of a variable whose nominal subtype is unconstrained, unless this subtype is indefinite, or the variable is aliased. A slice of an array shall not be renamed if this restriction disallows renaming of the array. In addition to the places

where Legality Rules normally apply, these rules apply also in the private part of an instance of a generic unit. These rules also apply for a renaming that appears in the body of a generic unit, with the additional requirement that even if the nominal subtype of the variable is indefinite, its type shall not be a descendant of an untagged generic formal derived type.

Discussion

The fix for example 1 is to forbid the renaming: even though the formal type looks indefinite, it is possible for the actual type to be definite. Note that the manual already covers the case where C1_Ren is declared in the visible part of the generic unit, because legality rules are checked in the instance (12.3(11)). We are extending the legality rules for object renaming to apply in the private part of the instance, and we are assuming the worst in the body.

To fix example 2, we could forbid a view conversion that obtains an indefinite view of an object whose nominal subtype is definite. However, such a view conversion was legal in Ada 83, so this would be an incompatibility. It seems better to mandate a check on the assignment to X: it is very surprising that this check is not there in the first place.

Also consider the following example:

```
package P is
   pragma Elaborate_Body;
   type T (D : Integer) is private;
private
   type T (D : Integer) is tagged
      record
         C : String (1 .. D);
      end record;
end P;

with P;
package Q is
   type NT (ND : Integer := 3) is new T (ND);
   X : NT;
   Y : NT (0);
end Q

with Q;
package body P is
   C_Ren : Character renames T (Q.X).C (2);
begin
   Q.X := Q.Y; -- Houston, we have a problem!
end P;
```

This is similar to example 2 above, except that here we don't use a view conversion to change the discriminant of Q.X. In this case the trouble comes from the view conversion T (Q.X) in the declaration of C_Ren. As long as all the types involved are tagged, renaming a component of a view conversion works fine, because tagged types don't have defaulted discriminants. But here we go through an untagged type to change the discriminants. It is clear that the conversion T (Q.X) should not be considered a view conversion.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0018**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **6.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0018 Full conformance of expressions with attributes

Working Reference Number AI95-00175

Question

Is Integer'Succ fully conformant with Integer'Pred? (No.)

>From 6.3.1(19-22), it would appear so: both attribute_references have the syntactic construction

prefix ' attribute_designator := name ' identifier

Summary of Response

For two attribute_references to fully conform, the attribute_designator must be the same.

Corrigendum Wording

Insert after 6.3.1(21):

- each direct_name, character_literal, and selector_name that is not part of the prefix of an expanded name in one denotes the same declaration as the corresponding direct_name, character_literal, or selector_name in the other; and

the new paragraph:

- each attribute_designator in one must be the same as the corresponding attribute_designator in the other; and

Discussion

It would be ludicrous to treat two different attributes to be fully conformant. None of the reasons for conformance checking would be enforced if this were true. Thus, the standard's failure to require this can only be categorized as an oversight.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0019**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **7.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0019 Delayed declaration of inherited primitive subprograms

Working Reference Number AI95-00033

Question

Is the rule given in 7.3.1(6) intended to apply to cases where a derived type is declared outside the declarative region in which its parent type is immediately declared? (No.)

Consider the following example:

```
package body R is

   package P is
      type Pt is ...;
   private
      procedure Op (X : Pt);
   end P;

   type T is new P.Pt;
      -- procedure Op (X : T); is inherited here but not yet declared

   package body P is
      -- procedure Op (X : T) is declared here, according to
      -- 7.3.1(6), because the corresponding declaration for Pt
      -- is visible at this point and the body of P is still within
      -- the immediate scope of T. It is somewhat strange, however,
      -- that the subprogram does not get declared immediately
      -- within the same declarative region as T.
      -- Is this the intent?  (No.)
   begin
      Op(T'(...)); -- Legal?  (No.)
   end P;

end R;
```

Also, are the rules of 7.3.1(3) and 7.3.1(4) regarding the availability of additional characteristics for composite types and derived types intended to apply when such types are declared outside the declarative region in which a component type or parent type is immediately declared? (No.)

Summary of Response

7.3.1 describes places where additional characteristics of a type become revealed. These rules apply only *immediately* within the declarative region in which the type is declared.

Corrigendum Wording

Replace 7.3.1(3):

> For a composite type, the characteristics (see 7.3) of the type are determined in part by the characteristics of its component types. At the place where the composite type is declared, the only characteristics of component types used are those characteristics visible at that place. If later within the immediate scope of the composite type additional characteristics become visible for a component type, then any corresponding characteristics become visible for the composite type. Any additional predefined operators are implicitly declared at that place.

by:

> For a composite type, the characteristics (see 7.3) of the type are determined in part by the characteristics of its component types. At the place where the composite type is declared, the only characteristics of component types used are those characteristics visible at that place. If later immediately within the declarative region in which the composite type is declared additional characteristics become visible for a

component type, then any corresponding characteristics become visible for the composite type. Any additional predefined operators are implicitly declared at that place.

Replace 7.3.1(4):

The corresponding rule applies to a type defined by a derived_type_definition, if there is a place within its immediate scope where additional characteristics of its parent type become visible.

by:

The corresponding rule applies to a type defined by a derived_type_definition, if there is a place immediately within the declarative region in which the type is declared where additional characteristics of its parent type become visible.

Replace 7.3.1(5):

For example, an array type whose component type is limited private becomes nonlimited if the full view of the component type is nonlimited and visible at some later place within the immediate scope of the array type. In such a case, the predefined "=" operator is implicitly declared at that place, and assignment is allowed after that place.

by:

For example, an array type whose component type is limited private becomes nonlimited if the full view of the component type is nonlimited and visible at some later place immediately within the declarative region in which the array type is declared. In such a case, the predefined "=" operator is implicitly declared at that place, and assignment is allowed after that place.

Replace 7.3.1(6):

Inherited primitive subprograms follow a different rule. For a derived_type_definition, each inherited primitive subprogram is implicitly declared at the earliest place, if any, within the immediate scope of the type_declaration, but after the type_declaration, where the corresponding declaration from the parent is visible. If there is no such place, then the inherited subprogram is not declared at all. An inherited subprogram that is not declared at all cannot be named in a call and cannot be overridden, but for a tagged type, it is possible to dispatch to it.

by:

Inherited primitive subprograms follow a different rule. For a derived_type_definition, each inherited primitive subprogram is implicitly declared at the earliest place, if any, immediately within the declarative region in which the type_declaration occurs, but after the type_declaration, where the corresponding declaration from the parent is visible. If there is no such place, then the inherited subprogram is not declared at all. An inherited subprogram that is not declared at all cannot be named in a call and cannot be overridden, but for a tagged type it is possible to dispatch to it.

Discussion

The wording of 7.3.1 was inherited from the Ada 83 standard's subsection 7.4.2, but by stating the rules in terms of the immediate scope of the type this inadvertently included all nested scopes, which was not intended.

Consider:

```
package Outer is
    package Inner is
        type Inner_Type is private;
    private
        type Inner_Type is new Boolean;
    end Inner;

    type Outer_Type is array(Natural range <>) of Inner.Inner_Type;
end Outer;

package body Outer is
    package body Inner is
        ...
        -- At this point, we can see that Inner_Type is a Boolean type.
```

```
            -- So does Outer_Type have an "and" operator here?  (No.)
        end Inner;
    end Outer;
```

The wording of 7.3.1(3) would seem to imply that an "and" operator is implicitly declared for type Outer_Type at the beginning of the body of Inner, since this is within the immediate scope of Outer_Type. However, in Ada 83, such an "and" operator was not implicitly declared -- such an operator could only be declared *immediately* within the declarative region of Outer_Type -- not in some nested Inner package.

This language change was not intended. Furthermore, the principle that implicit declarations of operators (or other additional characteristics) can only be revealed *immediately* within the declaration region of the outer type should be preserved, even in the case of new language features, such as child packages. Therefore, 7.3.1(3,4,5,6) are changed accordingly.

The Ada 83 standard prefaced paragraph 7.4.2(6) by saying, "If the composite type is itself declared within the package that declares the private type", which avoided the problems introduced by 7.3.1(3,4,6). In attempting to be more general and include derived types as well as composite types, plus handle the case of child units (which are not "within" their parent package but are "within the declarative region of" their parent), the restriction imposed by the Ada 83 preface was unintentionally lost. Note that the AARM does not list this as a "Change from Ada 83", which is further evidence that this change was not intended. Also, paragraph 7.3.1(7.b) of the AARM makes it clear that these rules were only meant to pertain to types declared within the same declarative region as the component type or parent type providing the additional operations.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0020**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **7.6; A.5.1; G.1.1; G.1.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0020 Classification of language-defined packages

Working Reference Number AI95-00126

Question

None of the language-defined packages has a pragma Remote_Types. This makes distributed programming less convenient, because the types declared in these packages cannot be transported across partitions (unless, of course, the package has a pragma Pure). Should some of the language-defined packages be remote types packages?

Summary of Response

The following language-defined packages are declared Pure:

Ada.Numerics.Complex_Elementary_Functions G.1.2(9)

Ada.Numerics.Complex_Types G.1.1(25)

Ada.Numerics.Elementary_Functions A.5.1(9)

The other nongeneric equivalents defined in these sections are also declared Pure, if they exist.

The following language-defined package is a remote types package:

Ada.Finalization 7.6(4)

For each language-defined generic package that is declared Pure (or preelaborated), the private part must not contain anything that would prevent instances from being declared Pure (preelaborated, respectively).

Corrigendum Wording

Replace 7.6(4):

```
package Ada.Finalization is
    pragma Preelaborate(Finalization);
```

by:

```
package Ada.Finalization is
    pragma Preelaborate(Finalization);
    pragma Remote_Types(Finalization);
```

Replace A.5.1(9):

The library package Numerics.Elementary_Functions defines the same subprograms as Numerics.Generic_Elementary_Functions, except that the predefined type Float is systematically substituted for Float_Type'Base throughout. Nongeneric equivalents of Numerics.Generic_Elementary_Functions for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Elementary_Functions, Numerics.Long_Elementary_Functions, etc.

by:

The library package Numerics.Elementary_Functions is declared pure and defines the same subprograms as Numerics.Generic_Elementary_Functions, except that the predefined type Float is systematically substituted for Float_Type'Base throughout. Nongeneric equivalents of Numerics.Generic_Elementary_Functions for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Elementary_Functions, Numerics.Long_Elementary_Functions, etc.

Replace G.1.1(25):

> The library package Numerics.Complex_Types defines the same types, constants, and subprograms as Numerics.Generic_Complex_Types, except that the predefined type Float is systematically substituted for Real'Base throughout. Nongeneric equivalents of Numerics.Generic_Complex_Types for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Types, Numerics.Long_Complex_Types, etc.

by:

> The library package Numerics.Complex_Types is declared pure and defines the same types, constants, and subprograms as Numerics.Generic_Complex_Types, except that the predefined type Float is systematically substituted for Real'Base throughout. Nongeneric equivalents of Numerics.Generic_Complex_Types for each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Types, Numerics.Long_Complex_Types, etc.

Replace G.1.2(9):

> The library package Numerics.Complex_Elementary_Functions defines the same subprograms as Numerics.Generic_Complex_Elementary_Functions, except that the predefined type Float is systematically substituted for Real'Base, and the Complex and Imaginary types exported by Numerics.Complex_Types are systematically substituted for Complex and Imaginary, throughout. Nongeneric equivalents of Numerics.Generic_Complex_Elementary_Functions corresponding to each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Elementary_Functions, Numerics.Long_Complex_Elementary_Functions, etc.

by:

> The library package Numerics.Complex_Elementary_Functions is declared pure and defines the same subprograms as Numerics.Generic_Complex_Elementary_Functions, except that the predefined type Float is systematically substituted for Real'Base, and the Complex and Imaginary types exported by Numerics.Complex_Types are systematically substituted for Complex and Imaginary, throughout. Nongeneric equivalents of Numerics.Generic_Complex_Elementary_Functions corresponding to each of the other predefined floating point types are defined similarly, with the names Numerics.Short_Complex_Elementary_Functions, Numerics.Long_Complex_Elementary_Functions, etc.

Discussion

Ada.Numerics has three children that are instances of declared-pure generic packages. The intent is that these instances also be declared pure, even though 8652/0034 (AI-00041) says that this does not happen by default.

Now, it might seem that many language-defined packages ought to be remote types packages. However, it turns out that this makes sense for very few language-defined packages, and is important for only one: Finalization. Detailed analysis follows.

The following is a list of all the language-defined library units. On the right, "Pure" or "Preelaborate" are the pragmas provided by the standard for each package. (The three instances mentioned above are also shown as "Pure".) For each package, "Yes" means it could reasonably be a remote types package; "No" means it should not. If neither "Yes" nor "No" is shown, that means "No", and the reason is that the type(s) declared in that package are inappropriate for transporting across partitions. This is true of the I/O packages, for example -- we don't want to require transporting files across partitions.

There is no point in making a declared-pure package into a remote types package, so all the ones marked "Pure" are "No" by default.

It makes sense for a package to have both pragmas Preelaborate and Remote_Types.

"No -- access type T" means, "We can't make it a remote types package, because it contains an access type, and we don't want that access type to be a remote access type."

"No -- depends on X" means, "We can't make it a remote types package, because it depends on X, and we have decided X should not be pure, shared passive, or remote types." See E.2.2(6).

Note that Ada.Calendar and Ada.Real_Time should not be remote types packages, because we wish to allow implementations to choose a different representation for the time-related types on different partitions. For example, type Calendar.Time on one partition might use a different "epoch" than on another partition. The types involved are: Calendar.Time, Real_Time.Time, Real_Time.Time_Span, and Real_Time.Seconds_Count. Thus, in order to pass time values across partitions, the programmer will have to define an application-specific time type, and translate to that.

Note that Ada.Exceptions is listed as "No", because it contains an access type that should not be a remote access type. It was one of the cases mentioned in the comments that prompted this issue. This is unfortunate; one would like to transport values of types Exception_Id and Exception_Occurrence.

Note that Ada.Task_Identification should not be a remote types package. If it were, then one could pass a Task_ID across partitions, and then do things like Abort_Task on that Task_ID on the "wrong" partition. This would require the tasking run-time system to know about distribution, which was never intended; alternatively, it would require us to declare such cases erroneous, which seems pointlessly error prone. Note also that this package cannot be declared Pure or Shared_Passive, because Task_ID is likely to be implemented using access types.

Note that the Discrete_Random and Float_Random packages are not made remote types packages. It makes no sense to copy the generators, since they are supposed to have reference semantics.

This leaves Finalization, Characters.Handling, Command_Line, and Interfaces.COBOL as potential candidates for being remote types packages. Of these, the only significant additional functionality is for Finalization. Therefore, we choose to make Finalization a remote types package, and leave the other three as specified in the current standard. (It seems silly, for example, to make Command_Line a remote types package, so that values of type Exit_Status can be transported, when Strings.Unbounded is not made a remote types package, so that values of type Unbounded_String cannot be transported; the latter would be far more useful, if it were possible.)

Analysis of each language-defined library unit follows:

```
Standard   A.1(4)                                                Pure
Ada   A.2(2)                                                    Pure
Ada.Asynchronous_Task_Control   D.11(3)
    No -- no need to make it a remote types package.
Ada.Calendar   9.6(10)
Ada.Characters   A.3.1(2)                                       Pure
Ada.Characters.Handling   A.3.2(2)                              Preelaborate
    Yes.
Ada.Characters.Latin_1   A.3.3(3)                               Pure
Ada.Command_Line   A.15(3)                                     Preelaborate
    Yes.
Ada.Decimal   F.2(2)                                            Pure
Ada.Direct_IO   A.8.4(2), A.9(3)
Ada.Dynamic_Priorities   D.5(3)
    No -- no need to make it a remote types package.
Ada.Exceptions   11.4.1(2)
    No -- access type Exception_Occurrence_Access.
Ada.Finalization   7.6(4)                                       Preelaborate
    Yes.
Ada.Float_Text_IO   A.10.9(32)
Ada.Float_Wide_Text_IO   A.11(3)
Ada.Integer_Text_IO   A.10.8(20)
Ada.Integer_Wide_Text_IO   A.11(3)
Ada.Interrupts   C.3.2(2)
    No -- access type Parameterless_Handler.
Ada.Interrupts.Names   C.3.2(12)
    No -- depends on Ada.Interrupts.
Ada.IO_Exceptions   A.13(3)                                     Pure
Ada.Numerics   A.5(3)                                           Pure
Ada.Numerics.Complex_Elementary_Functions   G.1.2(9)            Pure
Ada.Numerics.Complex_Types   G.1.1(25)                          Pure
Ada.Numerics.Discrete_Random   A.5.2(17)
```

```
    No -- the generator parameter is supposed to have reference semantics.
Ada.Numerics.Elementary_Functions   A.5.1(9)                Pure
Ada.Numerics.Float_Random   A.5.2(5)
    No -- the generator parameter is supposed to have reference semantics.
Ada.Numerics.Generic_Complex_Elementary_Functions   G.1.2(2) Pure
Ada.Numerics.Generic_Complex_Types   G.1.1(2)               Pure
Ada.Numerics.Generic_Elementary_Functions   A.5.1(3)        Pure
Ada.Real_Time   D.8(3)
Ada.Sequential_IO   A.8.1(2)
Ada.Storage_IO   A.9(3)                                     Preelaborate
    No -- depends on System.
Ada.Streams   13.13.1(2)                                    Pure
Ada.Streams.Stream_IO   A.12.1(3)
Ada.Strings   A.4.1(3)                                      Pure
Ada.Strings.Bounded   A.4.4(3)                              Preelaborate
    No -- depends on Ada.Strings.Maps.
    This means Bounded_Strings cannot be transported.
Ada.Strings.Fixed   A.4.3(5)                                Preelaborate
    No -- depends on Ada.Strings.Maps.
    This means the functions in this package cannot be called from
    the specification of a remote types package.
Ada.Strings.Maps   A.4.2(3)                                 Preelaborate
    No -- access type Character_Mapping_Function.
Ada.Strings.Maps.Constants   A.4.6(3)
    No -- depends on Ada.Strings.Maps.
Ada.Strings.Unbounded   A.4.5(3)
    No -- access type String_Access; depends on Ada.Strings.Maps.
    This means Unbounded_Strings cannot easily be passed across partitions.
Ada.Strings.Wide_Bounded   A.4.7(1)
    No -- depends on Ada.Strings.Wide_Maps.
Ada.Strings.Wide_Fixed   A.4.7(1)
    No -- depends on Ada.Strings.Wide_Maps.
Ada.Strings.Wide_Maps   A.4.7(3)
    No -- access type Character_Mapping_Function.
Ada.Strings.Wide_Maps.Wide_Constants   A.4.7(1)
    No -- depends on Ada.Strings.Wide_Maps.
Ada.Strings.Wide_Unbounded   A.4.7(1)
    No -- access type String_Access.
Ada.Synchronous_Task_Control   D.10(3)
    No -- no need to make it a remote types package.
Ada.Tags   3.9(6)
    No -- if type Tag needs to be transported, one can use the External_Tag
    function.
Ada.Task_Attributes   C.7.2(2)
    No -- access type Attribute_Handle.
Ada.Task_Identification   C.7.1(2)
Ada.Text_IO   A.10.1(2)
Ada.Text_IO.Complex_IO   G.1.3(3)
Ada.Text_IO.Editing   F.3.3(3)
Ada.Text_IO.Text_Streams   A.12.2(3)
Ada.Unchecked_Conversion   13.9(3)                          Pure
Ada.Unchecked_Deallocation   13.11.2(3)                     Preelaborate
    No -- a procedure cannot be categorized as a remote types library unit,
    by 8652/0078 (AI-00048).
Ada.Wide_Text_IO   A.11(2)
Ada.Wide_Text_IO.Complex_IO   G.1.4(1)
Ada.Wide_Text_IO.Editing   F.3.4(1)
Ada.Wide_Text_IO.Text_Streams   A.12.3(3)
Interfaces   B.2(3)                                         Pure
Interfaces.C   B.3(4)                                       Pure
Interfaces.C.Pointers   B.3.2(4)                            Preelaborate
    No -- access type Pointer.
Interfaces.C.Strings   B.3.1(3)                             Preelaborate
    No -- access type char_array_access.
Interfaces.COBOL   B.4(7)                                   Preelaborate
    Yes.
Interfaces.Fortran   B.5(4)                                 Pure
System   13.7(3)                                            Preelaborate
    No -- transportation of type Address makes no sense.
    However, note that an implementation is allowed to add pragma
    Remote_Types if it wants to.
System.Address_To_Access_Conversions   13.7.2(2)            Preelaborate
    No -- access type Object_Pointer.
System.Machine_Code   13.8(7)
```

```
        Don't care -- entire contents are implementation defined, so we don't
        need to say anything about this one.
    System.RPC   E.5(3)
        No -- RPC is used in the implementation of inter-partition communication,
        so it doesn't make sense to make it a remote types package.
    System.Storage_Elements   13.7.1(2)                          Preelaborate
        No -- depends on System.
    System.Storage_Pools   13.11(5)                              Preelaborate
        No -- depends on System.
```

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0021**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **7.6; 7.6.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0021 Extension aggregates with controlled subcomponents

Working Reference Number AI95-00182

Question

Question 1:

7.6(11) says: "For an extension_aggregate whose ancestor_part is a subtype_mark, Initialize is called on all controlled subcomponents of the ancestor part".

This seems inappropriate in the case of a controlled subcomponent for which a default initial value has been given. What is the intent?

Question 2:

7.6.1(13) says: "The anonymous objects created by function calls ... are finalized no later than the end of the innermost enclosing declarative_item or statement."

This rule permits a reference to a finalized object in the case where the function call is used as a name in an object renaming declaration:

```
X : Some_Controlled_Type renames Some_Function_Call;
```

and similarly when the function call is used as an actual parameter for a generic formal in out parameter, or when a component of the object returned by the function call is renamed.

Summary of Response

For an extension_aggregate whose ancestor_part is a subtype_mark, for each controlled subcomponent of the ancestor part, either Initialize is called, or the default initial value is assigned, as appropriate.

7.6.1(13) is modified so that an anonymous object is not finalized until after it is no longer accessible via any name.

Corrigendum Wording

Replace 7.6(11):

> For an extension_aggregate whose ancestor_part is a subtype_mark, Initialize is called on all controlled subcomponents of the ancestor part; if the type of the ancestor part is itself controlled, the Initialize procedure of the ancestor type is called, unless that Initialize procedure is abstract.

by:

> For an extension_aggregate whose ancestor_part is a subtype_mark, for each controlled subcomponent of the ancestor part, either Initialize is called, or its initial value is assigned, as appropriate; if the type of the ancestor part is itself controlled, the Initialize procedure of the ancestor type is called, unless that Initialize procedure is abstract.

Replace 7.6.1(13):

> The anonymous objects created by function calls and by aggregates are finalized no later than the end of the innermost enclosing declarative_item or statement; if that is a compound_statement, they are finalized before starting the execution of any statement within the compound_statement.

by:

> If the object_name in an object_renaming_declaration, or the actual parameter for a generic formal **in out** parameter in a generic_instantiation, denotes any part of an anonymous object created by a

function call, the anonymous object is not finalized until after it is no longer accessible via any name. Otherwise, an anonymous object created by a function call or by an aggregate is finalized no later than the end of the innermost enclosing declarative_item or statement; if that is a compound_statement, the object is finalized before starting the execution of any statement within the compound_statement.

Discussion

Question 1:

The intent is that Initialize should not be called when there is an initial value to be assigned.

Question 2:

The intent is that such renamed objects should not be finalized until they are no longer "in use".

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0022**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **7.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0022 Aggregates of a controlled type

Working Reference Number AI95-00083

Question

If an object of a controlled type is declared in the same package as the type, and initialized with an aggregate, is Program_Error raised? (No.)

Summary of Response

When an (extension) aggregate of a controlled type is assigned other than by an assignment or return statement, the aggregate is built "in place". No anonymous object is created and Adjust is not called on the target object.

Corrigendum Wording

Insert after 7.6(17):

> For an assignment_statement, after the name and expression have been evaluated, and any conversion (including constraint checking) has been done, an anonymous object is created, and the value is assigned into it; that is, the assignment operation is applied. (Assignment includes value adjustment.) The target of the assignment_statement is then finalized. The value of the anonymous object is then assigned into the target of the assignment_statement. Finally, the anonymous object is finalized. As explained below, the implementation may eliminate the intermediate anonymous object, so this description subsumes the one given in 5.2, ‘‘Assignment Statements’’.

the new paragraph:

> *Implementation Requirements*
>
> For an aggregate of a controlled type whose value is assigned, other than by an assignment_statement or a return_statement, the implementation shall not create a separate anonymous object for the aggregate. The aggregate value shall be constructed directly in the target of the assignment operation and Adjust is not called on the target object.

Response

When an aggregate of a controlled type is created and immediately assigned into an object other than in an assignment or return statement (that is in an initial expression, subaggregate, formal parameter, generic in parameter, or allocator), the implementation must not create a separate anonymous object for the aggregate; it must create the value of the aggregate directly in the target object. Thus, there is no assignment from the anonymous object to the target object, so the Finalize and Adjust that would be done for that assignment are not done.

Discussion

Consider the following controlled type:

```
type Dyn_String is private;
Null_String : constant Dyn_String;
...
private
type Dyn_String is new Ada.Finalization.Controlled
  with record
    ...
  end record;
procedure Finalize(X : in out Dyn_String);
procedure Adjust(X : in out Dyn_String);

Null_String : constant Dyn_String :=
  (Ada.Finalization.Controlled with ...);
```

Clearly, at the time when the full constant declaration for Null_String is elaborated, the bodies for Finalize and Adjust have not yet been elaborated. 7.6(21) gives the permission to build the aggregate directly in the target object, thereby eliminating the need for the assignment (and the associated calls on Adjust/Finalize):

> For an aggregate or function call whose value is assigned into a target object, the implementation need not create a separate anonymous object if it can safely create the value of the aggregate or function call directly in the target object.

However, it seems important to *require* this behavior, so that this kind of programming is portable (that is, it will work portably without raising Program_Error due to an access-before-elaboration from calling Adjust or Finalize before their bodies are elaborated).

In other words, the first sentence of 7.6(21) should be an implementation requirement in the case where a new object is being created.

Note that no Adjust ever takes place on an aggregate as a whole, since there is no assignment to the aggregate as a whole (4.3(5) and AARM 4.3(5.b)). AARM 7.6(21.a) talks about this case, and says that "only one value adjustment is necessary". This is misleading. It should say that only one adjustment of each controlled *subcomponent* (if any) is necessary in this case. *No* adjustments of the object as a whole are necessary (and as suggested above, such adjustments should be disallowed).

Note that this interpretation applies to all object creations, not just to object_declarations. Thus, continuing the above example, if we have:

```
type Dyn_String_Ptr is access all Dyn_String;
Null_String_Ptr: Dyn_String_Ptr :=
  new Dyn_String'(Ada.Finalization.Controlled with ...);
```

The aggregate must be built directly in the newly-created heap object.

Similarly, if we have

```
function Is_Null (Value : in Dyn_String) return Boolean;
```

then the aggregate actual parameter in the call

```
if Is_Null ((Ada.Finalization.Controlled with ...)) then
```

is built directly in a temporary object, and Adjust is not called on the object as a whole.

We exempt assignment and return statements from this requirement as there is no compelling reason to burden implementations with this requirement in those cases.

Note that all aggregates of a controlled type are extension aggregates: Controlled and Limited_Controlled are private, so it is not possible to create a normal record aggregate for such a type.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0023**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **7.6.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0023 Exceptions raised by Adjust/Finalize — missing cases

Working Reference Number AI95-00169

Question

7.6.1(14-20) list a variety of situations in which Finalize and Adjust may raise exceptions, and the possible consequences.

It does not seem to indicate what happens when an exception is raised in a Finalize which is part of the finalization of a master due to the most normal type of completion--reaching the end of the execution. What is the intended semantics for this situation? (Any pending finalizations are performed and Program_Error is raised.)

If the finalization of an anonymous object raises an exception what should occur? (Program_Error is raised at the point of finalization.)

If a transfer of control or raising of an exception occurs prior to performing a finalization of an anonymous object, when is the object finalized? (The anonymous object is finalized as part of the finalization of the innermost enclosing master.)

If an explicit call to Adjust or Finalize propagates an exception, is the exception converted to Program_Error? (No.)

Summary of Response

If a call to Finalize propagates an exception when invoked as part of the finalization of a master, Program_Error is raised at the point where normal execution would have continued following the master's finalization. Any other finalizations due to be performed up to that point are performed before raising Program_Error.

If a call to Finalize propagates an exception in the case of finalizing an anonymous object created for a function call or aggregate, Program_Error is raised at the point where normal execution would have continued following the object's finalization.

For finalizations of objects that occur as the result of transfers of control or the raising of an exception, the finalization of an anonymous object occurs as part of the finalizations due to be performed for the innermost enclosing master of the anonymous object.

For an explicit call to Adjust or Finalize that propagates an exception, the exception is propagated as for a normal call to a user-defined subprogram that propagates an exception.

Corrigendum Wording

Insert after 7.6.1(13):

> The anonymous objects created by function calls and by aggregates are finalized no later than the end of the innermost enclosing declarative_item or statement; if that is a compound_statement, they are finalized before starting the execution of any statement within the compound_statement.

the new paragraph:

> If a transfer of control or raising of an exception occurs prior to performing a finalization of an anonymous object, the anonymous object is finalized as part of the finalizations due to be performed for the object's innermost enclosing master.

Replace 7.6.1(14):

It is a bounded error for a call on Finalize or Adjust to propagate an exception. The possible consequences depend on what action invoked the Finalize or Adjust operation:

by:

It is a bounded error for a call on Finalize or Adjust that occurs as part of object finalization or assignment to propagate an exception. The possible consequences depend on what action invoked the Finalize or Adjust operation:

Insert after 7.6.1(17):

- For a Finalize invoked as part of a call on an instance of Unchecked_Deallocation, any other finalizations due to be performed are performed, and then Program_Error is raised.

the new paragraphs:

- For a Finalize invoked as part of the finalization of the anonymous object created by a function call or aggregate, any other finalizations due to be performed are performed, and then Program_Error is raised.
- For a Finalize invoked due to reaching the end of the execution of a master, any other finalizations associated with the master are performed, and Program_Error is raised immediately after leaving the master.

Discussion

The lack of a rule describing what happens when a call to Finalize propagates an exception during the finalization of a master is an oversight. The intended semantics is to treat this case similarly to what happens when Finalize propagates an exception when invoked by the transfer of control of a return statement (as defined by 7.6.1(18)). Any finalizations due to be performed are carried out and Program_Error is raised at the point where normal execution would have continued.

There is also no mention of what should happen when a call to Finalize propagates an exception when finalizing an anonymous object. Since such objects are not directly associated with a master, the rules of 7.6.1(14-20) don't appear to explain what should happen. The intended semantics is to raise Program_Error as in the case of other implicit calls to Finalize. The exception is raised immediately following the point where the Finalize operation is invoked (as defined by 7.6.1(13)). Also, in the presence of transfers of control or the raising of an exception, the finalization of anonymous objects occurs as part of the "finalizations due to be performed" mentioned in the rules of 7.6.1(18-19).

In the case of explicitly invoked Adjust and Finalize operations, any exception propagated by such calls should simply be propagated as for an exception propagation that occurs as part of a call to any other user-defined subprogram. There is no benefit in requiring such exceptions to be converted to Program_Error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0024**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **7.6.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0024 Initialize, Adjust, and exceptions

Working Reference Number AI95-00193

Question

If an object that is initialized by assignment fails during an Adjust operation, should the object nevertheless be finalized? (This is unspecified.)

Summary of Response

For a controlled object (including a component) that is initialized explicitly by assignment (possibly to an enclosing object), if its Adjust procedure is invoked but then fails by propagating an exception, it is not specified by the language whether the object is finalized. If the object is initialized by assignment from an aggregate, its Adjust procedure is not invoked (per 8652/0022 [AI-00083]), but it is finalized if and only if the initialization is successful. If it is initialized by assignment from something other than an aggregate, but its Adjust procedure is not invoked at all because initialization fails before that point, then the object is not finalized.

For an object that is initialized by default, the object is not finalized unless default initialization completes successfully, i.e. without propagating an exception.

For an Adjust invoked as part of initialization, if it propagates an exception, no other adjustments need be performed prior to raising Program_Error.

Corrigendum Wording

Replace 7.6.1(16):

- For an Adjust invoked as part of an assignment operation, any other adjustments due to be performed are performed, and then Program_Error is raised.

by:

- For an Adjust invoked as part of the initialization of a controlled object, other adjustments due to be performed might or might not be performed, and then Program_Error is raised. During its propagation, finalization might or might not be applied to objects whose Adjust failed. For an Adjust invoked as part of an assignment statement, any other adjustments due to be performed are performed, and then Program_Error is raised.

Response

7.6.1(4) says:

> ... each object ... is finalized if the object was successfully initialized and still exists.

This is relaxed for objects that are initialized by assignment when an Adjust propagates an exception. For such objects, the object may be finalized so long as the Adjust operation is invoked, even if it propagates an exception. It must be finalized if the Adjust operation completes successfully.

For objects which are initialized by default, no change in the wording is proposed; such objects are finalized if and only if default initialization completes without propagating an exception.

The definition of "adjustments due to be performed" should be relaxed for an assignment operation that is part of initialization, thereby allowing initialization to be abandoned as soon as any Adjust fails. For an assignment statement, it is important that all adjustments be performed, even if one fails, because all controlled subcomponents are going to be finalized.

Discussion

When an object (including a component) is initialized by an assignment other than from an aggregate, the Adjust operation is invoked. If this operation propagates an exception, then other Adjust operations that are already due to be performed are performed, and then Program_Error is raised.

What this means is that if you have a composite object which is initialized by an assignment from something other than an aggregate, and it has multiple controlled parts, then if one of the Adjust operations fails, the others are still invoked. Clearly those parts for which Adjust succeeds should be finalized per 7.6.1(4). However, 7.6.1(4) implies that the ones for which Adjust fails should not be finalized. However, for implementations that "bundle" all of the Adjust operations for all controlled parts of a composite type into a single "adjust-whole-object" procedure, it is burdensome to keep track of which parts failed and which succeeded, and only finalize those whose Adjust succeeded. Note that if some of the Adjust operations had failed in an assignment statement, all parts would ultimately still be finalized when the master is left.

One of the important goals of the finalization model with respect to exceptions (see 7.6.1(14-18)) is that if one controlled abstraction fails by raising an exception in Adjust or Finalize, this failure should not spread to other unrelated controlled abstractions. Even when one composite object happens to have two controlled parts, one from the "failed" abstraction and one from the "still-good" abstraction, the "still-good" abstraction should still have Adjust and Finalize called the appropriate number of times to keep reference counts correct, avoid dangling pointers, etc.

Given this goal, the "bundling" of Adjust operations, and the correspondence with assignment statements, it seems best to allow that, so long as the Adjust routine has been invoked on an object being initialized, Finalize may be invoked on the object.

On a somewhat separate issue, the notion of adjustments "due to be performed" (7.6.1(16)) need not apply to initialization by assignment. So long as a subcomponent is not going to be finalized, it need not be adjusted, even if it is initialized as part of an enclosing composite assignment operation for which some adjustments are performed. On the other hand, for an assignment that is part of an assignment statement, it is important that all adjustments be attempted, even if some of them fail, since all subcomponents are going to be finalized. This relaxation for adjustments that occur during initialization means that an initialization may be abandoned as soon as any Adjust fails, so long as those components which have never been adjusted are not finalized.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0025**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **8.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0025 Overriding by implicit declarations

Working Reference Number AI95-00044

Question

8.3(9-13) do not cover the case of an implicitly declared "/=" that corresponds to an explicitly declared "=" operator. Is it the intent that such a "/=" operator overrides a predefined "/="? (Yes.)

These paragraphs also fail to cover the case of a statement_identifier. Is it the intent that a statement_identifier overrides an inherited subprogram with the same name? (Yes.)

Summary of Response

For an explicit declaration of "=", the corresponding "/=" that is implicitly declared (if any) overrides the predefined "/=" (if any).

The implicit declaration of a statement_identifier overrides the implicit declaration of an inherited subprogram with the same identifier.

Corrigendum Wording

Replace 8.3(9):

Two homographs are not generally allowed immediately within the same declarative region unless one *overrides* the other (see Legality Rules below). A declaration overrides another homograph that occurs immediately within the same declarative region in the following cases:

by:

Two homographs are not generally allowed immediately within the same declarative region unless one *overrides* the other (see Legality Rules below). The only declarations that are *overridable* are the implicit declarations for predefined operators and inherited primitive subprograms. A declaration overrides another homograph that occurs immediately within the same declarative region in the following cases:

Replace 8.3(10):

- An explicit declaration overrides an implicit declaration of a primitive subprogram, regardless of which declaration occurs first;

by:

- A declaration that is not overridable overrides one that is overridable, regardless of which declaration occurs first;

Replace 8.3(26):

An explicit declaration is illegal if there is a homograph occurring immediately within the same declarative region that is visible at the place of the declaration, and is not hidden from all visibility by the explicit declaration. Similarly, the context_clause for a subunit is illegal if it mentions (in a with_clause) some library unit, and there is a homograph of the library unit that is visible at the place of the corresponding stub, and the homograph and the mentioned library unit are both declared immediately within the same declarative region. These rules also apply to dispatching operations declared in the visible part of an instance of a generic unit. However, they do not apply to other overloadable declarations in an instance; such declarations may have type conformant profiles in the instance, so long as the corresponding declarations in the generic were not type conformant.

by:

A non-overridable declaration is illegal if there is a homograph occurring immediately within the same declarative region that is visible at the place of the declaration, and is not hidden from all visibility by the non-overridable declaration. Similarly, the context_clause for a subunit is illegal if it mentions (in

a with_clause) some library unit, and there is a homograph of the library unit that is visible at the place of the corresponding stub, and the homograph and the mentioned library unit are both declared immediately within the same declarative region. These rules also apply to dispatching operations declared in the visible part of an instance of a generic unit. However, they do not apply to other overloadable declarations in an instance; such declarations may have type conformant profiles in the instance, so long as the corresponding declarations in the generic were not type conformant.

Discussion

For the "/=" issue, clearly it would be confusing if the predefined "/=" were visible instead of the one corresponding to the user-defined "=".

For the statement_identifier issue, clearly the statement_identifier should override, because although the declaration is implicit, the statement_identifier itself is sitting right there in the code. Furthermore, we don't want to have a case where a non-overloadable declaration is overloaded, which would be the case if the statement_identifier did not hide, and both were visible. This interpretation is also necessary for upward compatibility, because in Ada 83, a statement_identifier hides an inherited subprogram. This is illustrated by validation test B83033B.

In retrospect, it was probably a mistake to base the definition of overriding on whether or not a declaration is implicit. A better model might be as follows:

The implicit declaration of a predefined operator or an inherited subprogram is an "overridable declaration". [Only overridable declarations may be overridden.]

If two or more homographs occur immediately within the same declarative region, then:

1) at most one of them is allowed to be a non-overridable declaration;

2) a non-overridable declaration overrides an overridable declaration, independent of which comes first;

3) an inherited subprogram overrides a predefined operator, except for equality of tagged types, where the reverse applies;

4) for those pairs for which (1) to (3) don't apply, a later overridable declaration overrides an earlier one.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0026**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **8.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0026 Uniqueness of component names

Working Reference Number AI95-00150

Question

Consider the following example, inspired by validation test C730003:

```
package Parent is
   type T is tagged ...;
   type DT is new T with private;
private
   type DT is new T with record
      Y: ...;
   end record;
end Parent;

package Parent.Child is
   type DDT is new DT with record
      Y: ...; -- Legal?  (No.)
   end record;
end Parent.Child;
```

Both DT and DDT contain components called Y. Is this name duplication legal? (No.)

Summary of Response

A type extension is illegal if somewhere within its immediate scope it has two visible components with the same name. See also 8652/0102 (AI-00157).

Corrigendum Wording

Replace 8.3(26):

> An explicit declaration is illegal if there is a homograph occurring immediately within the same declarative region that is visible at the place of the declaration, and is not hidden from all visibility by the explicit declaration. Similarly, the context_clause for a subunit is illegal if it mentions (in a with_clause) some library unit, and there is a homograph of the library unit that is visible at the place of the corresponding stub, and the homograph and the mentioned library unit are both declared immediately within the same declarative region. These rules also apply to dispatching operations declared in the visible part of an instance of a generic unit. However, they do not apply to other overloadable declarations in an instance; such declarations may have type conformant profiles in the instance, so long as the corresponding declarations in the generic were not type conformant.

by:

> An explicit declaration is illegal if there is a homograph occurring immediately within the same declarative region that is visible at the place of the declaration, and is not hidden from all visibility by the explicit declaration. In addition, a type extension is illegal if somewhere within its immediate scope it has two visible components with the same name. Similarly, the context_clause for a subunit is illegal if it mentions (in a with_clause) some library unit, and there is a homograph of the library unit that is visible at the place of the corresponding stub, and the homograph and the mentioned library unit are both declared immediately within the same declarative region. These rules also apply to dispatching operations declared in the visible part of an instance of a generic unit. However, they do not apply to other overloadable declarations in an instance; such declarations may have type conformant profiles in the instance, so long as the corresponding declarations in the generic were not type conformant.

Discussion

3.4(14) says:

> Declarations of components, protected subprograms, and entries, whether implicit or explicit, occur immediately within the declarative region of the type, in the order indicated above, following the parent subtype_indication.

8.3(26) says:

> An explicit declaration is illegal if there is a homograph occurring immediately within the same declarative region that is visible at the place of the declaration, and is not hidden from all visibility by the explicit declaration.

It appears that this rule does not apply to the second declaration of Y, above, because the inherited Y is not visible at this place. However, the inherited Y later does become visible, since DDT is in a child package. This is intended to be illegal -- components are not intended to be overridable; nor are they allowed to be overloadable.

The problem is that 8.3(26) and other parts of 8.3 are based on whether certain declarations are implicit or explicit. This leads to the problem addressed by this issue. See also 8652/0025 (AI-00044), which addresses other problems with the same underlying cause.

The intent is that two or more homographs are not allowed immediately within the same declarative region, if there is a place where both are visible, unless all but one are overridden. (Note however, that this rule does not apply in instances!)

Note that if DDT were declared in another root library unit, rather than in a child of Parent, then the two Y's would be legal, since there would be no place where both are visible.

This problem can occur anytime components can become visible after the initial declaration. Consider:

```
package A is
    type Foo is tagged private;
    package B is
        type New_Foo is new Foo with record
           I: Integer; -- Illegal because Foo.I is visible in the body.
        end record; -- Foo.I is not visible here.
    end C;
private
    type Foo is  tagged record  I: Integer; end record;
end A;

package body A is
    package body B is
        -- Foo.I becomes visible here, but that means we have two components
        -- with the same name visible in same record.
    end C;
end B;
```

Because of this, the new rule simply says that it is illegal for two components with the same name to ever be visible.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0027**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **8.5.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0027 Circular renamings as body

Working Reference Number AI95-00135

Question

Consider the following example:

```
package Example is
   function F return Boolean;
   function G return Boolean renames F;
   function H return Boolean renames G;
private
   function F return Boolean renames H; -- Legal?  (No.)
end Example;
```

8.5.4(5) says:

> The profile of a renaming-as-body shall be subtype-conformant with that of the renamed callable entity, and shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the subprogram it declares takes its convention from the renamed subprogram; otherwise the convention of the renamed subprogram shall not be Intrinsic.

In the above example, the renaming-as-body for F occurs before F is frozen. Therefore, F takes its calling convention from H, which comes from G, which comes from F. So what is the calling convention of F?

Summary of Response

A circular renaming-as-body represents infinite recursion. If the renaming-as-body occurs before the subprogram whose body is being defined is frozen, the renaming-as-body is illegal.

Corrigendum Wording

Replace 8.5.4(5):

> The profile of a renaming-as-body shall be subtype-conformant with that of the renamed callable entity, and shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the subprogram it declares takes its convention from the renamed subprogram; otherwise the convention of the renamed subprogram shall not be Intrinsic.

by:

> The profile of a renaming-as-body shall be subtype-conformant with that of the renamed callable entity, and shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the subprogram it declares takes its convention from the renamed subprogram; otherwise the convention of the renamed subprogram shall not be Intrinsic. A renaming-as-body is illegal if the declaration occurs before the subprogram whose declaration it completes is frozen, and the renaming renames the subprogram itself, through one or more subprogram renaming declarations, none of whose subprograms has been frozen.

Insert after 8.5.4(8):

> For a call on a renaming of a dispatching subprogram that is overridden, if the overriding occurred before the renaming, then the body executed is that of the overriding declaration, even if the overriding declaration is not visible at the place of the renaming; otherwise, the inherited or predefined subprogram is called.

the new paragraph:

> *Bounded (Run-Time) Errors*

If a subprogram directly or indirectly renames itself, then it is a bounded error to call that subprogram. Possible consequences are that Program_Error or Storage_Error is raised, or that the call results in infinite recursion.

Response

For a subprogram whose body is defined by a renaming-as-body, if the rule in 8.5.4(5) requires the calling convention of the subprogram to be taken ultimately from itself, then the renaming-as-body is illegal.

Discussion

In the above example, the definition of the calling convention of F is not well-defined, because of the circular definition in 8.5.4(5). Therefore, we choose to make this case illegal.

An alternative would be to define the calling convention to be Ada in this case. However, the compiler needs to detect the circularity anyway, in order to avoid an infinite loop during semantic analysis. Therefore, we might as well let the compiler give an error message, rather than generating infinitely-recursive code.

Note that some circularities are legal. In particular, if the renaming-as-body completes a subprogram *after* that subprogram is frozen, the circularity is legal, and will be infinitely recursive at run time. For example:

```
package Pack_1 is
    procedure P;
end Pack_1;

package Pack_2 is
    procedure Q;
end Pack_2;

with Pack_2;
package body Pack_1 is
    procedure P renames Pack_2.Q;
end Pack_1;

with Pack_1;
package body Pack_2 is
    procedure Q renames Pack_1.P;
end Pack_2;
```

The above is legal, the convention of P and Q is Ada, and a call to P or Q will be infinitely recursive. Note that we don't want to make *this* case illegal, since it cannot be detected at compile time.

Here is another example of legal circularity:

```
type Ptr is access function return Integer:
function F return Integer;
P: Ptr := F'access;
function F return Integer renames P.all;
```

The convention of P.all is Ada, by 6.3.1(3). F is frozen by the declaration of P, by 13.14(6,4,11). Therefore, 8.5.4(5) does not specify the convention of F; it defaults to Ada, and the subtype conformance required by 8.5.4(5) is satisfied.

Any call to F or P.all will result in infinite recursion.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0028**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **8.5.4; A.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0028 Profile of predefined operators for scalar types

Working Reference Number AI95-00145

Question

4.5.2(8-9) say:

> The ordering operators are predefined for every specific scalar type T, and for every discrete array type T, with the following specifications:

```
function "<" (Left, Right : T) return Boolean
...
```

where the T is in italics. Similar definitions are given throughout section 4 for other predefined operators. What is the meaning of this italicized type name notation? Presumably, it is intended to refer to the base subtype, at least in some cases.

However, the predefined operators shown in package Standard do not always use the base subtype:

```
function "<" (Left, Right : Boolean) return Boolean; -- A.1(7)
function "<" (Left, Right : Integer'Base) return Boolean; -- A.1(15)
```

Summary of Response

The italicized T shown in the definitions of predefined operators means:

- T'Base, for scalars

- the first subtype, for tagged types

- the type without any constraint, in other cases

The definitions of the operators in section 4 take precedence over those shown in A.1 in package Standard; for example, the "<" operator on type Boolean has parameters of subtype Boolean'Base, not Boolean.

If a renaming-as-body completes a declaration before the subprogram it declares is frozen, then the profile of the renaming-as-body need not be subtype-conformant with that of the renamed callable entity. The profile of such a renaming-as-body must instead be mode conformant with that of the renamed callable entity.

Corrigendum Wording

Replace 8.5.4(5):

> The profile of a renaming-as-body shall be subtype-conformant with that of the renamed callable entity, and shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the subprogram it declares takes its convention from the renamed subprogram; otherwise the convention of the renamed subprogram shall not be Intrinsic.

by:

> The profile of a renaming-as-body shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the profile shall be mode-conformant with that of the renamed callable entity and the subprogram it declares takes its convention from the renamed subprogram; otherwise the profile shall be subtype-conformant with that of the renamed callable entity and the convention of the renamed subprogram shall not be Intrinsic.

Replace A.1(7):

```
          -- function "="   (Left, Right : Boolean) return Boolean;
```

```
        -- function "/="  (Left, Right : Boolean) return Boolean;
        -- function "<"   (Left, Right : Boolean) return Boolean;
        -- function "<="  (Left, Right : Boolean) return Boolean;
        -- function ">"   (Left, Right : Boolean) return Boolean;
        -- function ">="  (Left, Right : Boolean) return Boolean;
```

by:

```
        -- function "="   (Left, Right : Boolean'Base) return Boolean;
        -- function "/="  (Left, Right : Boolean'Base) return Boolean;
        -- function "<"   (Left, Right : Boolean'Base) return Boolean;
        -- function "<="  (Left, Right : Boolean'Base) return Boolean;
        -- function ">"   (Left, Right : Boolean'Base) return Boolean;
        -- function ">="  (Left, Right : Boolean'Base) return Boolean;
```

Replace A.1(9):

```
        -- function "and" (Left, Right : Boolean) return Boolean;
        -- function "or"  (Left, Right : Boolean) return Boolean;
        -- function "xor" (Left, Right : Boolean) return Boolean;
```

by:

```
        -- function "and" (Left, Right : Boolean'Base) return Boolean'Base;
        -- function "or"  (Left, Right : Boolean'Base) return Boolean'Base;
        -- function "xor" (Left, Right : Boolean'Base) return Boolean'Base;
```

Replace A.1(10):

```
        -- function "not" (Right : Boolean) return Boolean;
```

by:

```
        -- function "not" (Right : Boolean'Base) return Boolean'Base;
```

Discussion

Consider the following type declarations, where the comments show the subtypes associated with the italicized notation in section 4:

```
type T1 is range ...;                          -- T1'Base
type T2 is tagged ...;                         -- T2
type T3(D: Integer) is tagged ...;             -- T3
type T4 is array(Integer range <>) of Integer; -- T4
type T5 is array(Integer range 1..100)
   of Integer;                                 -- T5-without-the-constraint
type T6 is record ...;                         -- T6
```

Note that T2 and T6 are constrained, despite the fact that they have no constraint. Note also that in the case of T5, the subtype in question has no name in Ada, since the Base attribute is not defined for composite types.

The Boolean operators in A.1 are shown with the wrong subtypes -- Boolean'Base is correct.

Furthermore, 8.5.4(5) says:

> The profile of a renaming-as-body shall be subtype-conformant with that of the renamed callable entity, and shall conform fully to that of the declaration it completes. If the renaming-as-body completes that declaration before the subprogram it declares is frozen, the subprogram it declares takes its convention from the renamed subprogram; otherwise the convention of the renamed subprogram shall not be Intrinsic.

However, consider:

```
package P is
    type T is private;
private
    type T is new Integer'Base;
end P;
```

```
use P;

function Equals(X, Y: T) return Boolean;
function Equals(X, Y: T) return Boolean renames "=";
```

Without this ruling, the above renaming-as-body would be illegal, since it is not subtype conformant. However, if the full type declaration were "type T is new Integer;", then it would be legal. It is intolerable for the contents of the private part to affect the legality of a client in this way; therefore, we relax the rules for renamings-as-body that appear before the subprogram is frozen. Note that after the subprogram is frozen, one cannot use a renaming-as-body for a predefined operator, because it is intrinsic.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0029**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **9.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0029 Elaboration of a task with no task_definition

Working Reference Number AI95-00116

Question

A legal task_type_declaration is

```
task type TT;
```

By the grammar in 9.1(2), this task_type_declaration does not include a task_definition.

9.1(10) says that the elaboration of a task declaration elaborates the task_definition; what if there isn't one? (An empty task_definition is elaborated.)

9.1(11) says the elaboration of a task_definition creates the task type and its first subtype; if there is no task_definition, when are the task type and its first subtype created? (There is an empty task_definition.)

Summary of Response

For a task declaration with no task_definition, an empty task_definition is assumed.

Corrigendum Wording

Insert after 9.1(9):

> A task_definition defines a task type and its first subtype. The first list of task_items of a task_definition, together with the known_discriminant_part, if any, is called the visible part of the task unit. The optional list of task_items after the reserved word **private** is called the private part of the task unit.

the new paragraph:

> For a task declaration without a task_definition, a task_definition without task_items is assumed.

Discussion

The question not only applies to the syntax of 9.1(2) , but equally to the syntax of 9.1(3), i.e., to all task declarations.

The intent is clear. The new wording means that

```
task type TT;
```

is equivalent to

```
task type TT is
end TT;
```

providing an implicit, empty task_definition.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0030**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **9.6**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0030 Exception raised by Month, Day, Seconds in Ada.Calendar?

Working Reference Number AI95-00113

Question

In the Ada.Calendar package, the function Year and the procedure Split raise Time_Error if the given Date parameter represents a date outside the range 1901 .. 2099. What do the functions Month, Day, and Seconds do with such a date? (Raise Time_Error.)

Summary of Response

The functions Month, Day, and Seconds in Ada.Calendar raise Time_Error if the year is outside the range of the subtype Year_Number.

Corrigendum Wording

Replace 9.6(26):

> The exception Time_Error is raised by the function Time_Of if the actual parameters do not form a proper date. This exception is also raised by the operators "+" and "-" if the result is not representable in the type Time or Duration, as appropriate. This exception is also raised by the function Year or the procedure Split if the year number of the given date is outside of the range of the subtype Year_Number.

by:

> The exception Time_Error is raised by the function Time_Of if the actual parameters do not form a proper date. This exception is also raised by the operators "+" and "-" if the result is not representable in the type Time or Duration, as appropriate. This exception is also raised by the functions Year, Month, Day, and Seconds and the procedure Split if the year number of the given date is outside of the range of the subtype Year_Number.

Discussion

The implementation model for these functions is that the procedure Split is first called and then the required result is extracted. For example

```
function Month (Date : Time) return Month_Number is
   Y : Year_Number;
   M : Month_Number;
   D : Day_Number;
   S : Day_Duration;
begin
   Split(Date, Y, M, D, S);
   return M;
end Month;
```

If Split raises Time_Error then, by propagation, Month will also raise Time_Error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0031**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **9.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0031 Termination signals query of Terminate attribute

Working Reference Number AI95-00118

Question

Consider:

```
task body T is
   Stop_Pulse : Integer;

   task Local_Task is ... end Local_Task;

   task body Local_Task is
   begin
      Stop_Pulse := 17;
   end T1;

begin
   loop
      if Local_Task'Terminated then
         Rely_On(Stop_Pulse=17); -- Is this erroneous?  (No.)
         exit;
      end if;
   end loop;
end T;
```

Since there is no signaling as per 9.10(2-10) between the assignment to Stop_Pulse and any action in task T prior to the call on Rely_On, reliance on the update to Stop_Pulse by Local_Task is erroneous by 9.10(11). 9.10(6) doesn't apply, since T is not yet (or ever) waiting for the termination of Local_Task in the Ada technical sense of "waiting".

Summary of Response

A task T2 can rely on values of variables that are updated by another task T1, if task T2 first verifies that T1'Terminated is True.

Corrigendum Wording

Insert after 9.10(6):

- If A1 is part of the execution of a task, and A2 is the action of waiting for the termination of the task;

the new paragraph:

- If A1 is the termination of a task T, and A2 is either the evaluation of the expression T'Terminated or a call to Ada.Task_Identification.Is_Terminated with an actual parameter that identifies T (see C.7.1);

Discussion

It would be surprising if T'Terminated were True, but T failed to update any locally-cached variables, or the querying task failed to see those updates.

The wording is written so that Ada.Task_Identificiation.Is_Terminated(<expr>) (where <expr> evaluates to T'Identity) works the same as T'Terminated. It would be very surprising if this function, defined to be the same as 'Terminated, had a different signaling behavior.

Note that we do not say anything about the Callable attribute; if the Callable attribute becomes False, the task might still have a local cache that is inconsistent with global variables.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0032**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **10.1.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0032 A library subprogram_body should replace, not complete, an instance

Working Reference Number AI95-00192

Question

Can a library subprogram body ever complete an existing generic instance? (No.)

Summary of Response

A library subprogram body can be interpreted as the completion of a generic subprogram or of a subprogram that is not an instance of a generic subprogram but not as the completion of an instance of a generic subprogram.

Corrigendum Wording

Replace 10.1.4(4):

> If a library_unit_body that is a subprogram_body is submitted to the compiler, it is interpreted only as a completion if a library_unit_declaration for a subprogram or a generic subprogram with the same defining_program_unit_name already exists in the environment (even if the profile of the body is not type conformant with that of the declaration); otherwise the subprogram_body is interpreted as both the declaration and body of a library subprogram.

by:

> If a library_unit_body that is a subprogram_body is submitted to the compiler, it is interpreted only as a completion if a library_unit_declaration with the same defining_program_unit_name already exists in the environment for a subprogram other than an instance of a generic subprogram or for a generic subprogram (even if the profile of the body is not type conformant with that of the declaration); otherwise the subprogram_body is interpreted as both the declaration and body of a library subprogram.

Response

A library subprogram body never completes an existing generic instance, but replaces it.

Discussion

It is a general principle in Ada 95 (as in Ada 83) that a library generic instance be considered as a single lump and can not be considered decomposed into specification and body for the purposes of completion or replacement.

Suppose we compile

```
generic procedure GP;

procedure GP is ... end GP;

with GP;
procedure P is new GP;
```

and then submit

```
procedure P is
begin ... end P;
```

The consequence is that the newly submitted procedure P completely replaces the instance P. It does not act as a new completion for the instance specification and thereby just replace the notional body of the instance.

However, this intention was not clear in the Ada 83 standard but was clarified by AI83-199 and AI83-266. In particular, the latter says

"After instantiating a generic subprogram as a library unit, any attempt to compile a subprogram body having the same identifier as that of the library unit instantiation causes the instantiation to be deleted from the library and replaced with the new library unit subprogram."

It was the intention that the behaviour in Ada 95 be the same in this respect. However, there was a change of wording between Ada 83 and Ada 95 which might have been the source of confusion. In Ada 83 the term subprogram did not include an instance whereas in Ada 95 the term subprogram does include an instance.

In the Ada 95 standard, 10.1.4(4) says

"If a library_unit_body that is a subprogram_body is submitted to the compiler, it is interpreted only as a completion if a library_unit_declaration for a subprogram or a generic subprogram with the same defining_program_unit_name already exists in the environment ... "

This incorrectly uses the term "subprogram" where it intended to exclude the case of an instance and so should have said "a subprogram that is not an instance".

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0033**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **10.1.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0033 Placement of program unit pragmas in generic packages

Working Reference Number AI95-00136

Question

Consider:

```
generic
  pragma Pure; -- (a)  Legal? (No.)
  type F is . . .
package G is
  pragma Pure; -- (b)  Legal? (Yes.)
  type T is . . .
end G;
```

10.1.5(5) says the pragma shall appear:

> Immediately within the declaration of a program unit and before any nested declaration, in which case the argument, if any, shall be a direct_name that denotes the immediately enclosing program unit declaration.

This seems to imply that the pragma Pure belongs at (a), and not at (b). Is this the intent? (No.)

Consider also:

```
package P is
    -- No declarations here.
private
    pragma Pure; -- Legal? (No.)
end P;
```

Summary of Response

A program unit pragma for a generic package must appear at the beginning of the package specification, and not in the generic formal part. For any program unit, such a pragma must not appear in the private part.

Corrigendum Wording

Replace 10.1.5(5):

- Immediately within the declaration of a program unit and before any nested declaration, in which case the argument, if any, shall be a direct_name that denotes the immediately enclosing program unit declaration.

by:

- Immediately within the visible part of a program unit and before any nested declaration (but not within a generic formal part), in which case the argument, if any, shall be a direct_name that denotes the immediately enclosing program unit declaration.

Discussion

It was not the intent to allow or require a program unit pragma for a generic package at the beginning of the generic formal part. Furthermore, such a placement would be strange and confusing. Thus, the wording of 10.1.5(5) is incorrect in this case.

Likewise, it was not the intent to allow placement within a private part, just because there happen to be no declarations in the visible part. Allowing that would introduce a small but pointless implementation burden, and would be confusing, since Pure and so forth represent externally visible properties of program units.

Note that for a generic unit, the term "visible part" officially includes the generic formal part.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0034**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **10.1.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0034 Program unit pragmas in generic units

Working Reference Number AI95-00041

Question

Do program unit pragmas and in particular library unit pragmas within a generic unit and referring to the generic unit apply to all instances of the generic unit?

Consider:

```
generic
    ...
package P is
    pragma Pure(P);
 ...
end P;

package PI is new P(...);

generic
    ...
package Q is
    pragma Pure;
 ...
end Q;

package QI is new Q(...);
```

Does the pragma Pure apply to the respective instances PI and QI? (No.)

Since pragma Pure is a library unit pragma, are instantiations of P and Q illegal, if the resulting instances are not library units? (No.)

Summary of Response

Library unit pragmas within a generic unit and applying to the generic unit itself do not apply to instances of the generic unit, unless a specific rule of the pragma specifies the contrary.

If the user wants a library unit pragma without such a rule to apply to an instance, then that pragma must be repeated explicitly for the instance.

The following Implementation Advice is added: Program unit pragmas that are not library-unit pragmas, when supported for a generic unit, should apply to all instances of the generic for which there is not an overriding pragma applied directly to the instance.

Corrigendum Wording

Insert after 10.1.5(7):

> Certain program unit pragmas are defined to be *library unit pragmas*. The name, if any, in a library unit pragma shall denote the declaration of a library unit.

the new paragraphs:

> *Static Semantics*
>
> A library unit pragma that applies to a generic unit does not apply to its instances, unless a specific rule for the pragma specifies the contrary.
>
> *Implementation Advice*

> When applied to a generic unit, a program unit pragma that is not a library unit pragma should apply to each instance of the generic unit for which there is not an overriding pragma applied directly to the instance.

Response

Library unit pragmas within a generic unit and applying to the generic unit itself do not apply to instances of the generic unit, unless a specific semantic rule of the pragma specifies the contrary.

If the user wants such a pragma to apply to an instance, then it must be repeated explicitly for the instance.

The program unit pragma INLINE applies to all instances, based on an explicit semantic rule of the pragma. Since a ruling on the applicability of program unit pragmas affects only implementation-defined support of pragmas, an implementation advice should be added in 10.1.5 that, for program unit pragmas applied to generic units, the pragma should apply to all instances of the generic unit.

Discussion

An exegesis of the standard showed that a clear answer to the questions cannot be derived from it. This exegesis is not reproduced for the Defect Report, but can be retrieved from the appendix of the working document AI-00041.

An examination of the individual program unit pragmas follows.

Some general observations:

> 1. If a pragma that applies to a generic unit were not to apply to all instances, the user would still have the option to specify such a pragma for each of the instances individually. This may be cumbersome, but no language functionality is lost.
>
> 2. If a pragma that applies to a generic unit were to apply to all instances automatically, the user would lose the capability of specifying a pragma that applies to the generic unit only. Note that the user does not have the option of placing the pragma outside the generic package and thereby escape a "current instance rule" (discussed later) selectively, since such placement is not allowed for pragmas on generic packages (see 10.1.5(4)).

We now examine the pragma semantics for all program unit pragmas and discuss whether or not the pragma should apply to the generic unit only or to all its instances as well.

Pragma Preelaborate:

Consider the following example:

```
with user_defined_function;
generic
   ...
package P is
   pragma Preelaborate;
   X: integer := user_defined_function;
   ...
end P;
```

The generic unit is legal, since its elaboration does not call the imported function. (The purpose of the pragma is merely to ensure that the elaboration of the generic body occurs prior to the elaboration of all non-preelaborated library units; i.e., to avoid the need for elaboration checks upon instantiations).

Yet, if the pragma applies to the instances as well, they would all be illegal, since they are not preelaborable !

The semantics of pragma Preelaborate can been regarded as an expression of intent that the pragma not be automatically applicable to all instances. Otherwise the pragma should have enforced its restrictions on the nested declarations within the generic package to detect the above problem prior to any instantiation.

If pragma Preelaborate applied to all instances, the user would no longer have the means to force preelaboration of the body of the generic unit (as shown in the example) without also imposing such preelaboration requirements on all instances and restricting the instantiations to library level. (The semantics of the pragma when applied to local instances is somewhat ill-defined.)

We conclude that pragma Preelaborate should not automatically apply to all instances of the generic unit.

Pragma Pure:

Consider the following example:

```
generic
   type T is private;
package Q is
   pragma Pure(Q);
   type TN is new T;
   ...
end Q;
```

The user intention of pragma Pure in this package is to indicate that this generic unit can (but need not) be instantiated to yield a pure package. E.g.:

```
type Acc is access TT;
package Q_Acc is new Q(Acc);  -- not a pure package

package Q_Int is new Q(Integer); -- a pure package
pragma Pure(Q_Int);
```

If the pragma were applicable to all instances, the package Q_Acc would be illegal.

This would put the writer of reusable generic packages that satisfy the necessary conditions for pure instances into a serious dilemma: If the pragma Pure is inserted, the reusability is curtailed to pure instances only. If the pragma Pure is not inserted, reusability is curtailed, because instances can then not be pure, since the purity rules prohibit the necessary dependency on the generic unit.

This reuse problem arises not only from the nature of the actual parameters of the instantiation, but also (and more importantly) from the context clauses of the instantiation, i.e.,

```
with P;  -- P not pure
package Q_Int2 is new Q(P.T);  -- would be illegal irrespective
                               -- of the nature of P.T
```

and from any non-pure context of a local instantiation (if local instances were allowed at all, given that the pragma is a library unit pragma).

Among others, any predefined or implementation-defined pure generic packages could not be instantiated in any non-pure context, which would be a quite devastating consequence.

(A similar dilemma arises for all the other categorization pragmas when used (or not used) in reusable generic packages.)

We conclude that pragma Pure definitely should not automatically apply to all instances of the generic unit.

Pragmas Shared_Passive and Remote_Types:

These pragmas impose the necessary restrictions to create a shared passive or a remote types library unit, respectively. As in the case of pragma Pure, such a pragma for a generic unit is a precondition for any instantiation to be so classified. However, unlike pragma Pure, it is not quite as obvious why instances should not be automatically in the respective category as well.

In the case of shared passive packages there is a potential issue when an application may be required to execute in environments that may or may not support storage nodes. If an instantiation is shared passive then the library unit may only be assigned to a single partition. In environments that do not support storage nodes this may be unnecessarily restrictive since it is possible that a non-categorized instance can be replicated in different partitions without compromising execution (e.g., if there are no variables in the package specification since a typical use of shared passive packages may be to store constant data that are common to different partitions).

In the case of remote types packages, one may wish to declare a type with subprograms that may be accessed both locally and remotely depending upon the instantiation. If the pragma applies to all instantiations then, when any declared access type within the package is referenced, it must be treated as containing a potentially remote access value. (This is particularly relevant, if the implementation uses wide pointers to represent remote access type values.)

One might perhaps surmise that despite the above scenarios, user convenience might argue for automatically applying these pragmas to all instances. However, we observe that no semantic difficulties arise if this were not the case and that such implicit "inheritance" seems contrary to the principle that critical specifications should be explicit. The cited situations show that to a-priori preclude that reusable packages can be instantiated both in restricted and unrestricted contexts may be unwise.

Pragma All_Calls_Remote, Remote_Call_Interface:

It is not unreasonable for applications to develop generic packages that may be instantiated to provide both locally and remotely accessible subprograms. For example, consider the case of a partition that provides the same interface for both intra-partition and inter-partition clients. If an instantiation is always a remote interface package, then intra-partition clients will incur the cost of calling subprograms through a compiler generated stub.

Additionally, a confirmation of the intent that this pragma should be explicitly specified is present in AARM E.2.3(15.b) where it is stated, "We considered making the public child of an RCI package implicitly RCI, but it seemed better to require an explicit pragma to avoid any confusion." It seems inconsistent to require an explicit pragma for a public child and not require an explicit pragma for an instantiation.

We conclude that the pragma should be applicable to the generic unit only.

Pragma Elaborate_Body:

Pragma Elaborate_Body applied to generic library unit forces the elaboration of the body of the unit immediately after the elaboration of the generic declaration. In the case of instantiations, this effect of the pragma merely restates the existing rules on instance elaboration, 12.3(20), so that this effect is the rule for instantiations in general. Hence, there is no need to make the pragma apply automatically to all instances, while it would be most detrimental to enforce as a secondary consequence that such instantiations yield only library units.

We conclude that the pragma should be applicable to the generic unit only.

Pragmas Elaborate and Elaborate_All:

These pragmas are irrelevant for this discussion, as they refer to program units other than the (generic) unit in which they appear. (Consequently, the affected units are elaborated prior to the elaboration of the generic unit and, hence, its instances.)

Pragma Inline:

Pragma Inline (historically) subscribes to the rule that it applies to all instances, when given for a generic subprogram. Its application to all instances relies on explicitly stated semantics of the pragma given in 6.3.2(5).

Pragmas Convention, Export, Import:

The applicability of these program unit pragmas to generic units is left implementation-defined by this International Standard. Thus, implementations can choose whichever semantics seem most appropriate. Since these pragmas are not library unit pragmas, inheritance of the pragma by instances of the generic unit does not have the detrimental effects shown earlier in this analysis. In fact, if the pragmas were to apply only to the generic unit and not to its instances, it would be difficult to associate any semantics with them. The most natural interpretation is therefore that the pragmas apply to all instances of the generic unit.

This concludes the list of predefined program unit pragmas. We have seen that, in some cases, applicability of the pragma to all instances would be seriously detrimental. We have seen other cases of library unit pragmas, where applicability to all instances may be more convenient on occasion, but is neither absolutely necessary nor warrants a rule that *a priori* precludes reusable generic units that can be instantiated in both restricted and unrestricted contexts.

Finally, we have seen that the existing language-defined program unit pragmas that are not library unit pragmas should apply to their instances. For the Inline pragma, this rule is already explicitly stated. However, as this presently matters only in cases, where applicability of the pragma to generic units is implementation-defined, and one can equally well conceive of future language-defined or implementation-defined pragmas, where automatic applicability to instances would not be appropriate, it was decided to make such an inheritance of pragmas by instances merely implementation advice, not a general semantic rule.

Although a program unit pragma on a generic should then generally be "inherited" by its instances, it might be overridden by a pragma applied directly to the instance, e.g., by a Convention or Export pragma. This is analogous to the rule for inheriting representation items by a derived type from its parent type. The inherited specification may be overridden by a direct specification on the derived type itself.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0035**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **10.2.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0035 Subunits of a preelaborated subprogram

Working Reference Number AI95-00002

Question

10.2.1(11) says, "All compilation units of a preelaborated library unit shall be preelaborable." The term "all compilation units" includes subunits. Is this really intended? (No.)

Summary of Response

A subunit which is not elaborated as part of elaborating a preelaborated library unit need not be preelaborable.

Corrigendum Wording

Replace 10.2.1(11):

> If a pragma Preelaborate (or pragma Pure — see below) applies to a library unit, then it is *preelaborated*. If a library unit is preelaborated, then its declaration, if any, and body, if any, are elaborated prior to all non-preelaborated library_items of the partition. All compilation units of a preelaborated library unit shall be preelaborable. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit. In addition, all compilation units of a preelaborated library unit shall depend semantically only on compilation units of other preelaborated library units.

by:

> If a pragma Preelaborate (or pragma Pure — see below) applies to a library unit, then it is *preelaborated*. If a library unit is preelaborated, then its declaration, if any, and body, if any, are elaborated prior to all non-preelaborated library_items of the partition. The declaration and body of a preelaborated library unit, and all subunits that are elaborated as part of elaborating the library unit, shall be preelaborable. In addition to the places where Legality Rules normally apply (see 12.3), this rule applies also in the private part of an instance of a generic unit. In addition, all compilation units of a preelaborated library unit shall depend semantically only on compilation units of other preelaborated library units.

Discussion

It is unnecessary to require a subunit of a preelaborated subprogram to be preelaborable, because such a subunit is not elaborated during elaboration of the subprogram. This also applies to subunits of preelaborated tasks and any other unit which does not elaborate its contents when it is elaborated.

Furthermore, subunits and physically nested program units should behave in the same way. If a subprogram is preelaborated, packages physically nested within the subprogram need not be preelaborable; therefore, the same should be true of package subunits.

Note that a subunit of a preelaborated package is required to be preelaborable even without the quoted sentence, because such a subunit is elaborated during elaboration of the parent package, and the definition of preelaborability in 10.2.1(5) says, "... unless its elaboration performs...", which makes the rule transitive.

The proposed change to the rule makes it always the case that subunits and physically nested program units behave the same, even in the face of nesting or multiple levels of subunits.

Note that the rule is irrelevant for subunits that are subprograms, since subprograms are always preelaborable. But it is relevant for package and other subunits.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0036**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **11.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0036 Access_Check is performed for access discriminants

Working Reference Number AI95-00176

Question

Is the null check that occurs when evaluating a discriminant association for an access discriminant considered to be an Access_Check? (Yes.)

Summary of Response

The check that an access discriminant is non-null is an Access_Check.

Corrigendum Wording

Replace 11.5(11):

Access_Check
: When evaluating a dereference (explicit or implicit), check that the value of the name is not **null**. When passing an actual parameter to a formal access parameter, check that the value of the actual parameter is not **null**.

by:

Access_Check
: When evaluating a dereference (explicit or implicit), check that the value of the name is not **null**. When passing an actual parameter to a formal access parameter, check that the value of the actual parameter is not **null**. When evaluating a discriminant_association for an access discriminant, check that the value of the discriminant is not **null**.

Discussion

It was an oversight to omit the null check that occurs on discriminant association for an access discriminant from the list of checks associated with Access_Check in 11.5(11).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0037**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **12.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0037 Predefined operators for generic formal array types

Working Reference Number AI95-00043

Question

There is an inconsistency between paragraphs 7.3.1(3) and 12.5(8) regarding predefined operators for formal array types in some rare cases. The former says that additional predefined operators may be declared when additional characteristics of the component type become known, whereas 12.5(8) says that all predefined operators are declared immediately after the formal type declaration.

Example:

```
package P is
   pragma Elaborate_Body;   -- just to make its body legal

   type Pt is private;

   generic
      type Ft is array( 1 .. 9 ) of Pt;
   package G is
   end G;

private
   type Pt is new Boolean;
end P;

package body P is

   package body G is
      -- relational operators declared for Ft here? (Yes.)
      -- 12.5(8) says no, 7.3.1(3) says yes
   end G;

end P;
```

(A similar example could be made using a public generic child unit to P, in which case the relational operators would be declared when entering the private part of the public child).

Summary of Response

For a generic formal type whose properties depend on a partial view (for example, a formal array type whose component type is a private type) the rules of 7.3.1 apply. Thus, the primitive subprograms of the formal type are not necessarily declared immediately after its declaration.

Corrigendum Wording

Replace 12.5(8):

> The formal type also belongs to each class that contains the determined class. The primitive subprograms of the type are as for any type in the determined class. For a formal type other than a formal derived type, these are the predefined operators of the type; they are implicitly declared immediately after the declaration of the formal type. In an instance, the copy of such an implicit declaration declares a view of the predefined operator of the actual type, even if this operator has been overridden for the actual type. The rules specific to formal derived types are given in 12.5.1.

by:

> The formal type also belongs to each class that contains the determined class. The primitive subprograms of the type are as for any type in the determined class. For a formal type other than a formal derived type, these are the predefined operators of the type. For an elementary formal type, the predefined operators are implicitly declared immediately after the declaration of the formal type. For a composite formal type, the predefined operators are implicitly declared either immediately after the declaration of the formal

type, or later in its immediate scope according to the rules of 7.3.1. In an instance, the copy of such an implicit declaration declares a view of the predefined operator of the actual type, even if this operator has been overridden for the actual type. The rules specific to formal derived types are given in 12.5.1.

Response

For a generic formal type whose properties depend on a partial view (for example, a formal array type whose component type is a private type) the rules of 7.3.1 apply. Thus, the primitive subprograms of the formal type are not necessarily declared immediately after its declaration.

Discussion

7.3.1(3) and 12.5(8) are in conflict for generic formal types. 7.3.1(3) should take precedence, since otherwise the privacy of private types would be violated. Furthermore, this interpretation is compatible with Ada 83.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0038**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **12.5.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0038 Primitives of formal type derived from another formal type

Working Reference Number AI95-00202

Question

In an instance of a generic with a formal derived type whose ancestor type is another formal type, the rules regarding the meanings of the implicit declarations for the formal derived type produce a peculiar result.

Consider the following example:

```
package P1 is
   type R1 is record ... end record;
   procedure S (x : R1);                  -- [1]
end P1;
use P1;

generic
   type F2 is new R1;
   -- implicit: procedure S (x : F2);     -- [2]
   type F3 is new F2;
   -- implicit: procedure S (x : F3);     -- [3]
procedure G (o2 : F2; o3 : F3);
procedure G (o2 : F2; o3 : F3) is
begin
   S(o2);
   S(o3);        -- Peculiar result: Calls S [5] in instance I?  (No.)
end G;

package P2 is
   type R2 is new R1;
   -- implicit: procedure S (x : R2);     -- [4]
   procedure S (x : out R2);              -- [5] Overriding with mode out
end P2;

package P3 is
   type R3 is new P2.R2;
   -- implicit: procedure S (x : R3);     -- [6]
   procedure S (x : R3);                  -- [7]
end P3;

procedure I is new G (P2.R2, P3.R3);
```

In the instance I, the implicit declarations of S which operate on F2 and F3, respectively, are the corresponding primitive subprograms of the ancestor types of each type, as stated in 12.5.1(21). The ancestor type of F2 is R1, so the implicit declaration of S that operates on F2 [2] is a view of the corresponding primitive subprogram of R1 [1].

The ancestor type of F3 is the type of the subtype denoted by the name F2 in the instance, which is R2. So, the implicit declaration of S that operates on F3 [3] is a view of the corresponding primitive subprogram of R2 [5]. But, the annotation in AARM 12.5.1(21.a) indicates that the reason the primitives of a formal derived type in an instance are views of its ancestor's primitives is because the primitives of its actual type might not be subtype conformant with those of its ancestor type. This intention could be violated if the primitive S [3] is a view of the primitive S [5].

Is it the intent that the primitive S [3] should declare a view of S [1]? (Yes.)

In general, when the ancestor type of a formal derived type is itself another formal type, then within an instance does the derived type acquire the primitive operations of the formal ancestor type or the primitive operations of the ancestor type's corresponding actual type? (The primitive operations of the formal ancestor.)

Summary of Response

In an instance of a generic unit having a formal derived type whose ancestor is itself a formal type, the copies of the implicit subprogram declarations of the formal derived type declare views of the corresponding copies of the primitive subprograms of the formal ancestor type.

Corrigendum Wording

Replace 12.5.1(21):

> For a formal derived type, the predefined operators and inherited user-defined subprograms are determined by the ancestor type, and are implicitly declared at the earliest place, if any, within the immediate scope of the formal type, where the corresponding primitive subprogram of the ancestor is visible (see 7.3.1). In an instance, the copy of such an implicit declaration declares a view of the corresponding primitive subprogram of the ancestor, even if this primitive has been overridden for the actual type. In the case of a formal private extension, however, the tag of the formal type is that of the actual type, so if the tag in a call is statically determined to be that of the formal type, the body executed will be that corresponding to the actual type.

by:

> For a formal derived type, the predefined operators and inherited user-defined subprograms are determined by the ancestor type, and are implicitly declared at the earliest place, if any, within the immediate scope of the formal type, where the corresponding primitive subprogram of the ancestor is visible (see 7.3.1). In an instance, the copy of such an implicit declaration declares a view of the corresponding primitive subprogram of the ancestor of the formal derived type, even if this primitive has been overridden for the actual type. When the ancestor of the formal derived type is itself a formal type, the copy of the implicit declaration declares a view of the corresponding copied operation of the ancestor. In the case of a formal private extension, however, the tag of the formal type is that of the actual type, so if the tag in a call is statically determined to be that of the formal type, the body executed will be that corresponding to the actual type.

Response

The copies of the implicitly declared primitive subprograms of a formal derived type in an instance are defined to be views of the ancestor type's corresponding operations (12.5.1(21)). In the case of a formal type whose ancestor is another formal type of the same generic unit, this results in the undesirable semantics that in an instance, the copies of the first formal type's implicitly declared operations are views of the corresponding operations of the ancestor type's actual type.

It is essential to ensure that such copied implicit operations are always views of some ancestor known at the point of the generic formal type's declaration, since using the operations of the formal ancestor's actual type can lead to inconsistencies because the operations of an actual (untagged) type may not be subtype conformant with those of the formal type.

The rule of 12.5.1(21) is amended to correct this problem. The copies of a formal derived type's operations in an instance are defined to be views of the corresponding copies of the primitive operations of the formal type's ancestor when the ancestor is a formal type, rather than simply those of "the ancestor type" (which in an instance would denote the actual type associated with the formal type's ancestor).

Note that in the case where the formal ancestor type is a formal derived type, the copied operations of the ancestor type in the instance are themselves views of operations coming from the ancestor type's own ancestor (so the new rule applies transitively for arbitrary levels of derivation from formal derived types).

Discussion

12.5.1(21) defines the implicit operations that are declared for a formal derived type, as well as the meaning of the copies of those implicit operations declared within an instance. The second sentence states:

> In an instance, the copy of such an implicit declaration declares a view of the corresponding primitive subprogram of the ancestor, even if this primitive has been overridden for the actual type.

However, in the instance of a generic unit with formal derived type T2 whose ancestor type is itself a formal derived type T1, the phrase "of the ancestor" must be interpreted as referring to the actual type A1 associated with the formal ancestor T1. This is because in the instance, the ancestor type of the copy of T2 is a view of the actual type A1 associated with T1. This is the normal interpretation of copies of declarations within instances as defined by the static semantics in 12.3(13-16). But that leads to the conclusion that the view defined in 12.5.1(21) denotes the corresponding primitive subprogram of the ancestor's actual type.

As shown in the example of the question section, this can result in inconsistent views of a formal type's primitive operations, since the formal view of a primitive may not be subtype conformant with the view in an instance. For example, modes of parameters may differ between the formal and actual views of an subprogram, leading to undefined semantics for the copied version of a call to such a subprogram from within an instance. This would essentially result in a generic contract model violation in the body of the instance.

The Ramification in AARM-12.5.1(21.a) makes the intent behind 12.5.1(21) clear, explaining how in the case of untagged types the rule ensures that the operations of the type in an instance are those of the ancestor rather than those of the actual type, which may not be subtype conformant. However the formulation of the rule does not account for cases where the ancestor is a formal type itself, whose operations may not be subtype conformant with those of a corresponding actual type in an instance.

This problem is fixed by specifying that, in an instance, the implicit declaration of a primitive subprogram of a formal derived type with a formal ancestor declares a view of the corresponding copied operation of the ancestor.

If the ancestor is a nonderived formal type, then the copied operations of the ancestor declare views of the predefined operators of the ancestor's corresponding actual type.

In the case where the ancestor is itself a formal derived type, then the copied operations of the ancestor will themselves be views of operations coming from the ancestor type's own ancestor, so the rule applies transitively for arbitrary levels of derivation from formal derived types.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0039**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **12.7**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0039 Formal object matching for formal packages

Working Reference Number AI95-00213

Question

For a generic formal object of mode in, the rule in 12.7(6) defines a matching rule for actuals of generic formal packages. If the type of such an object is a type which cannot have static expressions (such as a tagged type), can the actuals ever match? (Yes.)

An actual for an instance denotes a new stand-alone constant initialized by the actual to the instantiation, as described in 12.4(10). Therefore, the actuals for a formal package can never statically denote the same constant. Is this correct? (No.)

Here is an example:

```
package Pack is
  type Count_Type is tagged record;
    Count : Integer := 0;
  end record;

  TC_Default_Count : constant Count_Type := (Count => 0);
end Pack;

generic
  type Item (<>) is tagged private;
  TC_Default_Value : Item;
package Test_0 is
...
end Test_0;

with Test_0;
generic
  type Item_Type (<>) is tagged private;
  Default : Item_Type;
  with package Stacker is new Test_0 (Item_Type, Default);
procedure Test_1 (S : in out Stacker.Stack; I : in Item_Type);

with Pack;
with Test_0;
pragma Elaborate (Test_0);
package Test_2 is new Test_0 (Pack.Count_Type,
  Pack.TC_Default_Count);

procedure Test is
  package Count_Stacks renames Test_2;
  procedure TC_Count_Test is new Test_1 (Pack.Count_Type,
            Pack.TC_Default_Count,Count_Stacks); -- Legal? (Yes.)
  ...
end Test;
```

Note that Test_2.TC_Default_Value denotes a constant initialized by Pack.TC_Default_Count, while TC_Count_Test.Default denotes Pack.TC_Default_Count. Do these match by the rule of 12.7(6)? (Yes.)

Summary of Response

For a generic formal object of mode in, the rule in 12.7(6) is applied to the actual parameter of the actual instance, and the actual parameter for the formal package. If the actual parameter for the formal package is itself a formal parameter (of another generic unit), the actual for that parameter is used for matching. The latter rule is applied recursively.

Corrigendum Wording

Insert after 12.7(8):

- For other kinds of formals, the actuals match if they statically denote the same entity.

the new paragraph:

> For the purposes of matching, any actual parameter that is the name of a formal object of mode **in** is replaced by the formal object's actual expression (recursively).

Response

The intent of the standard is that actuals are always used for the matching rule of 12.7(6). In addition, the intent is that formals denote the associated actual for the purposes of the matching rule of 12.7(6).

Discussion

The intent of the standard is that actual objects are always used for the matching rule of 12.7(6). In addition, the intent is that formal objects used as actual parameters are ignored for the purposes of the matching rule of 12.7(6). Using formal objects as the actual parameter for a formal package is a natural way to use formal packages. In addition, all existing Ada compilers already support matching ignoring formal objects (as one of the validation tests required this behavior).

If the strict language in the standard was followed, this example would be illegal. Indeed, any use of a formal object as an actual for a formal object of mode in to a formal package would be illegal by 12.7(6). This would require a substantial restructuring of formal packages if adding a formal object was necessary. In some cases, no workaround is available. In addition, existing code may depend on this feature, since compilers have supported it for at least four years.

The revised rule is written in terms of eliminating all formal objects used directly as actuals, in order to avoid confusion. Other rules about which formal objects are eliminated could be considered (only local objects, only a single object, etc.) but these do not make the language easier to implement, just more confusing for the user.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0040**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **13.1; 13.3; 13.13.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0040 Inheritance of stream attributes for type extensions

Working Reference Number AI95-00108

Question

13.1(15) says:

> A derived type inherits each type-related aspect of its parent type that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent type from the grandparent type. A derived subtype inherits each subtype-specific aspect of its parent subtype that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent subtype from the grandparent subtype, but only if the parent subtype statically matches the first subtype of the parent type. An inherited aspect of representation is overridden by a subsequent representation item that specifies the same aspect of the type or subtype.

Do these rules apply to the stream-oriented attributes Read, Write, Input, and Output? (No.)

If an untagged derived type includes a known discriminant part, the number of discriminants can change. If we inherit the parent's attribute definition, we could write the wrong number of discriminants. Consider:

```
type Parent (D1, D2 : Integer := 1) is ...;
type Child (D : Integer := 2) is new Parent (D1 => D, D2 => D);
```

Clearly the default implementation of Parent'Write writes two discriminant values. How many discriminants does Child'Write write? (One.)

Are the stream-oriented attributes intended to work properly for language-defined types such as Unbounded_String? (Yes.)

Summary of Response

For a type extension, the predefined Read attribute is defined to call the Read of the parent type, followed by the Read of the non-inherited components, if any, in canonical order. The analogous rule applies to the Write attribute.

The Input and Output attributes are not inherited by a type extension.

Default stream attributes are never inherited; rather, the default implementation for the derived type is used.

The stream attributes must work properly for every language-defined nonlimited type. For language-defined private types, the output generated by the Write attribute is not specified, but it must be readable by the Read attribute.

Corrigendum Wording

Replace 13.1(15):

> A derived type inherits each type-related aspect of its parent type that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent type from the grandparent type. A derived subtype inherits each subtype-specific aspect of its parent subtype that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent subtype from the grandparent subtype, but only if the parent subtype statically matches the first subtype of the parent type. An inherited aspect of representation is overridden by a subsequent representation item that specifies the same aspect of the type or subtype.

by:

A derived type inherits each type-related aspect of representation of its parent type that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent type from the grandparent type. A derived subtype inherits each subtype-specific aspect of representation of its parent subtype that was directly specified before the declaration of the derived type, or (in the case where the parent is derived) that was inherited by the parent subtype from the grandparent subtype, but only if the parent subtype statically matches the first subtype of the parent type. An inherited aspect of representation is overridden by a subsequent representation item that specifies the same aspect of the type or subtype.

In contrast, whether operational aspects are inherited by a derived type depends on each specific aspect. When operational aspects are inherited by a derived type, aspects that were directly specified before the declaration of the derived type, or (in the case where the parent is derived) that were inherited by the parent type from the grandparent type are inherited. An inherited operational aspect is overridden by a subsequent operational item that specifies the same aspect of the type.

Insert after 13.1(18):

- If an aspect of representation of an entity is not specified, it is chosen by default in an unspecified manner.

the new paragraph:

If an operational aspect is *specified* for an entity (meaning that it is either directly specified or inherited), then that aspect of the entity is as specified. Otherwise, the aspect of the entity has the default value for that aspect.

Replace 13.3(75):

S'External_Tag

S'External_Tag denotes an external string representation for S'Tag; it is of the predefined type String. External_Tag may be specified for a specific tagged type via an attribute_definition_clause; the expression of such a clause shall be static. The default external tag representation is implementation defined. See 3.9.2 and 13.13.2.

by:

S'External_Tag

S'External_Tag denotes an external string representation for S'Tag; it is of the predefined type String. External_Tag may be specified for a specific tagged type via an attribute_definition_clause; the expression of such a clause shall be static. The default external tag representation is implementation defined. See 3.9.2 and 13.13.2. The value of External_Tag is never inherited; the default value is always used unless a new value is directly specified for a type.

Replace 13.13.2(9):

For elementary types, the representation in terms of stream elements is implementation defined. For composite types, the Write or Read attribute for each component is called in a canonical order. The canonical order of components is last dimension varying fastest for an array, and positional aggregate order for a record. Bounds are not included in the stream if *T* is an array type. If *T* is a discriminated type, discriminants are included only if they have defaults. If *T* is a tagged type, the tag is not included.

by:

For untagged derived types, the Write and Read attributes of the parent type are inherited as specified in 13.1; otherwise, the default implementations of these attributes are used. The default implementations of Write and Read attributes execute as follows:

For elementary types, the representation in terms of stream elements is implementation defined. For composite types, the Write or Read attribute for each component is called in canonical order, which is last dimension varying fastest for an array, and positional aggregate order for a record. Bounds are not included in the stream if *T* is an array type. If *T* is a discriminated type, discriminants are included only if they have defaults. If *T* is a tagged type, the tag is not included. For type extensions, the Write or Read attribute for the parent type is called, followed by the Write or Read attribute of each component of the extension part, in canonical order. For a limited type extension, if the attribute of any ancestor type of *T*

has been directly specified and the attribute of any ancestor type of the type of any of the extension components which are of a limited type has not been specified, the attribute of *T* shall be directly specified.

Replace 13.13.2(25):

Unless overridden by an attribute_definition_clause, these subprograms execute as follows:

by:

For untagged derived types, the Output and Input attributes of the parent type are inherited as specified in 13.1; otherwise, the default implementations of these attributes are used. The default implementations of Output and Input attributes execute as follows:

Replace 13.13.2(36):

The stream-oriented attributes may be specified for any type via an attribute_definition_clause. All nonlimited types have default implementations for these operations. An attribute_reference for one of these attributes is illegal if the type is limited, unless the attribute has been specified by an attribute_definition_clause. For an attribute_definition_clause specifying one of these attributes, the subtype of the Item parameter shall be the base subtype if scalar, and the first subtype otherwise. The same rule applies to the result of the Input function.

by:

The stream-oriented attributes may be specified for any type via an attribute_definition_clause. All nonlimited types have default implementations for these operations. An attribute_reference for one of these attributes is illegal if the type is limited, unless the attribute has been specified by an attribute_definition_clause or (for a type extension) the attribute has been specified for an ancestor type. For an attribute_definition_clause specifying one of these attributes, the subtype of the Item parameter shall be the base subtype if scalar, and the first subtype otherwise. The same rule applies to the result of the Input function.

Implementation Requirements

For every subtype S of a language-defined nonlimited specific type *T*, the output generated by S'Output or S'Write shall be readable by S'Input or S'Read, respectively. This rule applies across partitions if the implementation conforms to the Distributed Systems Annex.

Discussion

The general rule for inheritance of type-related representation aspects should not apply to the stream attributes of type extensions. For 'Read and 'Write, a rule analogous to the rule for tagged equality makes the most sense. For 'Input and 'Output, no inheritance makes sense; instead, they should regain their predefined meaning in terms of 'Read and 'Write.

There are several problems associated with applying the normal 13.1(15) inheritance rules to the stream attributes of tagged types:

1) Inheriting a 'Read or 'Write of the parent type as-is for the 'Read or 'Write of a type extension will ignore any new components added in the extension part. A rule analogous to the one for the equality operator makes more sense. In particular, the default 'Read or 'Write for a type extension should be defined to do the 'Read or 'Write of the parent type followed by the 'Read or 'Write for each component of the type extension, in canonical order.

2) Inheriting a 'Input or 'Output of the parent type as-is for 'Input or 'Output of a type extension makes no sense, since the inherited 'Input is a function returning the parent type, and the inherited 'Output puts out the discriminants of the parent type. For these two, the only meaningful approach seems to be for the default 'Input and 'Output for a tagged type to always be defined in terms of the 'Read and 'Write for the tagged type, preceded with the discriminants, if any.

For untagged derived types, there is no (new) problem for the derived type inheriting the stream attributes. Even for tagged derived types, if the extension part is null, the 'Read and 'Write will effectively be inherited.

We must take care, however, that all of the components have the appropriate attributes. For a limited type extension, the extension component could be of a type that does not have an implementation of Write or Read. In that case, we must take care to insure that the attribute for the new type does handle the extension component. We do this by requiring an attribute to be directly specified if it has a limited extension component that does not have an implementation of Write or Read and the parent type has a (specified) implementation of Write or Read. (An alternative would be to inherit the original operation unmodified, but this would silently ignore the extension components. This could cause hard-to-find bugs as the components would probably revert to default values when they are input.) This rule is similar to the way that functions of type extensions are inherited: they aren't inherited, they must be overridden (except that we only invoke it when we can't do the right thing automatically, which minimizes the places where existing code becomes illegal).

To see how this works in practice, consider the following example:

```
   package P is
      type T is limited tagged ...;
      for T'Read use ...;

      type Der is new T with null record;  --  OK (no extension components,
                                           --  T'Read is effectively inherited)

      type Der_Int is new T with -- OK (non-limited extension components,
                                 --  T'Read is inherited with the additional
                                 --  components added)
         record
            Int : Integer;
         end record;

      protected type Protect_Type is ... -- Note: no 'Read specified.

      type Der_Protect_Type is new T with -- Illegal unless
                                    -- Der_Protect_Type'Read is specified;
                                    -- we can't compose T'Read, as
                                    -- Protect_Type'Read can't be called.
         record
            PT : Protect_Type;
         end record;
   end P;
```

Simply making the operation uncallable doesn't work, as the operations can be dispatched. If, in the above example, Der_Protect_Type couldn't be called, problems would arise if T'Class'Read was called on a Der_Protect_Type object.

Clearly, the properties of the default implementation for the stream attributes can change for derived types (as in the example given in the question). Thus, we always want to use a "fresh" default implementation for an attribute, rather than inheriting a default implementation from the parent type.

For language-defined nonlimited private types, the International Standard does not say whether the stream-oriented attributes must work properly. It seems that they ought to. For many such types, the default version will work properly. However, for a type like Unbounded_String, which is almost certainly implemented as a data structure involving access values, the default versions will not work. Therefore, for these types, the implementer must provide an explicit version of the Read and Write attributes.

The wording takes advantage of the newly defined "operational attributes" (see 8652/0009 [AI-00137]) to say whether operational attributes are inherited depends on the attribute. This simplifies the wording by eliminating the need to describe a long list of exceptions to an inheritance rule that we want only in some cases, and provides future flexibility.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0041**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **13.11**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0041 Incorrect syntax in example — remove "limited"

Working Reference Number AI95-00066

Question

The syntax of the example 13.11(39) appears incorrect. Is it wrong? (Yes.)

Summary of Response

The reserved word "limited" should be removed from the example in 13.11(39).

Corrigendum Wording

Replace 13.11(39):

```
type Mark_Release_Pool_Type
    (Pool_Size : Storage_Elements.Storage_Count;
     Block_Size : Storage_Elements.Storage_Count)
         is new Root_Storage_Pool with limited private;
```

by:

```
type Mark_Release_Pool_Type
    (Pool_Size : Storage_Elements.Storage_Count;
     Block_Size : Storage_Elements.Storage_Count)
         is new Root_Storage_Pool with private;
```

Discussion

This syntax was left from a previous draft of the standard.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0042**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **13.12; D.7; H.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0042 Enforcing Restrictions might violate the generic contract model

Working Reference Number AI95-00130

Question

H.4(8) says:

> No_Local_Allocators
>
> > Allocators are prohibited in subprograms, generic subprograms, tasks and entry bodies; instantiations of generic packages are also prohibited in these contexts.

Why are instantiations prohibited in these contexts? (This ruling allows them.)

The restrictions No_Task_Hierarchy and No_Nested_Finalization do not prohibit such instantiations.

Summary of Response

The enforcement of restrictions might violate the contract model of generics, as well as violate the "privateness" of code in a private part or body.

To be consistent with the No_Task_Hierarchy and No_Nested_Finalization restrictions, the No_Local_Allocators restriction should not preclude nested generic instantiations. No_Nested_Finalization is broadened to cover finalization associated with protected and task objects. For the purposes of these (post-compilation) rules, a generic template is logically expanded at the point of each instantiation, and all of the expressions of the type definition for a record or protected type logically appear at the point of any default-initialized object creation, and default parameter expressions logically appear where used.

Corrigendum Wording

Insert after 13.12(8):

> A pragma Restrictions is a configuration pragma; unless otherwise specified for a particular restriction, a partition shall obey the restriction if a pragma Restrictions applies to any compilation unit included in the partition.

the new paragraphs:

> For the purpose of checking whether a partition contains constructs that violate any restriction (unless specified otherwise for a particular restriction):
>
> - Generic instances are logically expanded at the point of instantiation;
> - If an object of a type is declared or allocated and not explicitly initialized, then all expressions appearing in the definition for the type and any of its ancestors are presumed to be used;
> - A default_expression for a formal parameter or a generic formal object is considered to be used if and only if the corresponding actual parameter is not provided in a given call or instantiation.

Insert after 13.12(9):

> An implementation may place limitations on the values of the expression that are supported, and limitations on the supported combinations of restrictions. The consequences of violating such limitations are implementation defined.

the new paragraph:

> An implementation is permitted to omit restriction checks for code that is recognized at compile time to be unreachable and for which no code is generated.

Replace D.7(4):

No_Nested_Finalization
Objects with controlled parts and access types that designate such objects shall be declared only at library level.

by:

No_Nested_Finalization
Objects with controlled, protected, or task parts, and access types that designate such objects, shall be declared only at library level.

Replace H.4(8):

No_Local_Allocators
Allocators are prohibited in subprograms, generic subprograms, tasks, and entry bodies; instantiations of generic packages are also prohibited in these contexts.

by:

No_Local_Allocators
Allocators are prohibited in subprograms, generic subprograms, tasks, and entry bodies.

Response

An implementation supporting the No_Task_Hierarchy, No_Nested_Finalization, or No_Local_Allocators restrictions must enforce the intent of these restrictions with checks prior to run-time.

For the purposes of these checks:

- Generic instances are logically expanded at the point of instantiation;

- If an object of a type is declared or allocated and not explicitly initialized, then all expressions appearing in the definition for the type and any of its ancestors are presumed to be used.

- Default formal parameters are presumed to be used only if the corresponding actual parameter is not provided in a given call or instantiation;

- Notwithstanding the above, for code which is recognized at compile-time as unreachable, and for which no object code is generated, implementations are permitted to omit these checks.

No_Task_Hierarchy means that only tasks directly dependent on the master representing the execution of the environment task (body) are permitted. Tasks dependent on masters which correspond to other bodies or blocks are not permitted, even if these masters are executed by the environment task.

No_Nested_Finalization should be broadened to mean that objects requiring finalization due to having a controlled, protected, or task part are not permitted unless they are at the library level.

No_Local_Allocators means that allocators are prohibited in subprograms, generic subprograms, task bodies, and entry bodies. As indicated above, rather than precluding nested instantiations, instantiations are to be logically expanded at the point of instantiation for the purposes of this check.

Discussion

Precluding nested generic instances for the No_Local_Allocators restriction in H.4(8) in an attempt to preserve a generic contract model for restrictions is inconsistent with the rules for No_Task_Hierarchy given in D.7(3) and for No_Nested_Finalization given in D.7(4). In general, enforcing pragma Restrictions across a partition will necessarily violate the "privateness" of a private part or a body, as well as the generic contract model.

Although it might be useful to know that if a generic body does not by itself violate a restriction, then neither will any instantiation, enforcing this kind of "contract" rule for restrictions that distinguish library level from non-library level usages would overly limit the nested instantiations of useful, benign generics. Furthermore,

the pragma Restrictions is primarily designed to support application environments where schedulability and formal verification requirements dictate that generics can only be certified with respect to particular instantiations.

A more serious problem with the rules given for the No_Task_Hierarchy restriction in D.7(3), No_Nested_Finalization in D.7(4), and No_Local_Allocators in H.4(8), is that they do not properly account for violations appearing in expressions used for default parameters and for default initialization.

For example, the following partition obeys the static criteria given in H.4(8) for the No_Local_Allocators restriction, yet (in the absence of code-removing optimizations) the main subprogram evaluates an allocator:

```
package P is
   type Integer_Pointer is access Integer;
   type R is
      record
         C: Integer_Pointer := new Integer;
      end record;
end P;

with P;
procedure Main is
   X: P.R;  -- X.C is initialized by invoking an allocator
begin
   null;
end Main;
```

To close such loopholes, it is necessary to logically substitute default initialization and default parameters in line at the point of usage.

To ease the burden for implementations which check these restrictions late in the compilation process or during linking, implementations are permitted to omit the checks within constructs which generate no object code, because they are recognized as unreachable. Code which presumes a given implementation takes advantage of this permission is clearly less portable.

For an implementation that shares code between generic instantiations, it might be necessary for it to collect information while compiling a generic body which would allow the implementation to determine at link-time whether particular instantiations do or do not violate these restrictions. This is similar to other information gathering that is required of all implementations as part of implementing the Restrictions pragma, so this is not felt to be unduly burdensome.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0043**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **13.12**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0043 Compile-time enforcement of pragma Restrictions

Working Reference Number AI95-00190

Question

Shall we allow implementations to reject (i.e. refuse to run) programs that have run-time detected violations of pragma Restrictions, when they are detectable at compile time? (Yes.)

Summary of Response

Whenever enforcement of a restriction imposed by pragma Restrictions is not required by the standard prior to execution, but left to implementation-defined behaviour of dynamic semantics, it is reasonable to interpret pre-execution enforcement as a valid implementation-defined behaviour, provided that every execution of the partition will violate the restriction.

Corrigendum Wording

Insert after 13.12(9):

> An implementation may place limitations on the values of the expression that are supported, and limitations on the supported combinations of restrictions. The consequences of violating such limitations are implementation defined.

the new paragraph:

> Whenever enforcement of a restriction is not required prior to execution, an implementation may nevertheless enforce the restriction prior to execution of a partition to which the restriction applies, provided that every execution of the partition would violate the restriction.

Discussion

For the particularly critical rejection of programs that violate restrictions imposed by pragma Restrictions, the standard provides for implementation-defined behaviour in lieu of a compile- or link-time check otherwise required by 13.12(8). It is reasonable to interpret pre-execution enforcement of a configuration pragma as a valid implementation-defined behaviour, even if such enforcement is not required to occur prior to execution. This is particularly true for D.7(15), as this clause recommends the raising of a Storage_Error exception but does not specify the place where such an exception is to be raised.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0044**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **13.13.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0044 Components of Stream_Element_Array should be aliased

Working Reference Number AI95-00181

Question

13.13.1(4) does not define the components of Streams.Stream_Element_Array to be aliased. However, this makes various uses of this type inconvenient. What is the intent?

Summary of Response

The components of Streams.Stream_Element_Array are aliased:

```
type Stream_Element_Array is
   array(Stream_Element_Offset range <>) of aliased Stream_Element;
                                         ^^^^^^^
```

The following implementation permission is added:

> If Stream_Element'Size is not a multiple of System.Storage_Unit, then the components of Stream_Element_Array need not be aliased.

Corrigendum Wording

In 13.13.1(4) replace:

```
type Stream_Element_Array is
    array(Stream_Element_Offset range <>) of Stream_Element;
```

by:

```
type Stream_Element_Array is
    array(Stream_Element_Offset range <>) of aliased Stream_Element;
```

Insert after 13.13.1(9):

> The Write operation appends Item to the specified stream.

the new paragraph:

> *Implementation Permissions*
>
> If Stream_Element'Size is not a multiple of System.Storage_Unit, then the components of Stream_Element_Array need not be aliased.

Response

On machines where it is feasible, the components should be aliased.

Discussion

On machines where it is feasible, one should be allowed to form access values pointing to any component within a Stream_Element_Array. However, this package was intentionally designed to support machines where the addressable unit is different from the unit of storage used by the "network". Therefore, the requirement to be aliased is relaxed on such machines. Programmers wishing to write code that is portable to such machines should not take advantage of the aliased components.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0045**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.13.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0045 Exception raised at end of stream

Working Reference Number AI95-00132

Question

Suppose one gets a stream from the Ada.Text_IO.Streams.Stream function. (The same question applies to the Wide_Text_IO version, and also to streams created by Ada.Streams.Stream_IO.)

What happens if the stream's position corresponds to end-of-file, and one tries to get an item using the default version of T'Read, for some type T? Is Data_Error or End_Error raised? Can the result be abnormal?

Summary of Response

If the default version of T'Read (for some type T) is used to read from a stream, then if end of stream is encountered, End_Error is raised.

Corrigendum Wording

Insert after 13.13.2(35):

> In the default implementation of Read and Input for a composite type, for each scalar component that is a discriminant or whose component_declaration includes a default_expression, a check is made that the value returned by Read for the component belongs to its subtype. Constraint_Error is raised if this check fails. For other scalar components, no check is made. For each component that is of an access type, if the implementation can detect that the value returned by Read for the component is not a value of its subtype, Constraint_Error is raised. If the value is not a value of its subtype and this error is not detected, the component has an abnormal value, and erroneous execution can result (see 13.9.1).

the new paragraph:

> In the default implementation of Read and Input for a type, End_Error is raised if the end of the stream is reached before the reading of a value of the type is completed.

Discussion

13.13.1(8) admits to the notion of "end of stream" for stream types:

> The Read operation transfers Item'Length stream elements from the specified stream to fill the array Item. The index of the last stream element transferred is returned in Last. Last is less than Item'Last only if the end of the stream is reached.

The Read operation does not raise any exception at end of stream -- it just indicates this fact in the value returned in Last. Of course, what constitutes end of stream is defined by the particular stream type. The stream returned by Text_Streams.Stream has a notion of end of stream that corresponds to the end of the text file. A user-defined stream might have a different notion of end of stream, or might not have any such notion -- it is quite possible to implement a stream type that represents an infinitely long sequence.

So the question is, if a given stream type has a notion of end of stream, then what happens when T'Read hits the end (which it can detect by looking at the Last parameter returned by Streams.Read)?

A.13(13,17) say:

> The exception Data_Error can be propagated by the procedure Read (or by the Read attribute) if the element read cannot be interpreted as a value of the required subtype. ...
>
> If the element read by the procedure Read (or by the Read attribute) cannot be interpreted as a value of the required subtype, but this is not detected and Data_Error is not propagated, then the resulting value can be abnormal, and subsequent references to the value can lead to erroneous execution, as explained in 13.9.1.

Note that it is somewhat odd that the Read attribute can raise IO_Exceptions.Data_Error, since streams have nothing directly to do with I/O, and a given invocation of the Read attribute does not know whether it is dealing with a file stream or not. Nonetheless, that's what it says. Raising IO_Exceptions.End_Error is no more or less odd in this regard.

Possibilities are:

Alternative 1: Either Data_Error is raised, or an abnormal value is returned. This alternative is supported by the wording of A.13(13,17). Reading zero bytes, or an insufficient number of bytes, clearly gives a malformed piece of data. The programmer is forced to encode the number of elements in the stream somehow, or otherwise encode the end of stream explicitly, in order to avoid erroneous execution.

Alternative 2: Data_Error is raised. There is really no implementation reason for allowing abnormal values, since the condition is easy to detect, and a very minor efficiency hit. However, this alternative still forces the programmer to encode the end of stream by hand, since Data_Error does not distinguish between malformed data and end of stream.

Alternative 3: End_Error is raised. This alternative *still* forces the programmer to encode the end of stream by hand, because it does not distinguish between encountering the end of the stream in between stream elements, versus in the middle of an element -- the latter being a case of malformed data.

Alternative 4: End_Error is raised if the programmer calls the T'Read and the stream is at end of stream. However, Data_Error is raised if end of stream is encountered in the middle. This allows the programmer to reliably read a sequence of items from a stream, and notice when the last item has been read (by detecting End_Error), and distinguish this situation from malformed data (Data_Error). Thus, the programmer does not need to add extra data to the stream to explicitly encode end of stream. However, this alternative is harder to implement, since Read attributes are highly recursive. For example, suppose T is a record type with two components. If 'Read raises End_Error on the second component, T'Read must catch that exception, and turn it into Data_Error -- the second component wasn't malformed, but the record as a whole *is* malformed. On the other hand, an End_Error raised by reading the first component would simply be propagated by T'Read. In addition, if the user-defined overriding of the Read attribute would presumably want to mimic this behavior. (Note: AI83-00307 requires a similar behavior for Get procedures in Text_IO.)

Alternatives 5,6,7,8: Same as alternatives 1,2,3,4, but define Data_Error and/or End_Error exceptions in Streams, rather than using the ones from IO_Exceptions. This might be more elegant, but serves no practical purpose, and is too big a change to make at this point.

We choose Alternative 3, because it seems the friendliest alternative that has a reasonable implementation cost.

The programmer can reliably detect end-of-file for file streams as follows:

```
if not End_Of_File(An_Input_File) then
   T'Read(Stream(An_Input_File), Value);
   ...
end if;
```

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0046**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **13.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0046 Freezing rules

Working Reference Number AI95-00106

Question

1. Does an object_renaming_declaration cause freezing of the renamed object? (Yes.)

For example, is the following legal? (No.)

```
package P is
    type T is private;
    type Acc is access T;
    X: Acc;
    Y: T renames X.all; -- Illegal!
private
    type T is ...;
end P;
```

---

2. Now, consider the following example:

```
package P is
    type T(D: Integer) is private;
    type A is access T;
    Obj: A;
    I: Integer := Obj.D; -- Does this freeze T? (Yes,
                         -- and it's therefore illegal.)
private
    type T(D: Integer) is ...;
end P;
```

Does the declaration of I freeze the type T? (Yes.) If we replaced "Obj.D" with "Obj.all.D", then it would freeze T, and therefore be illegal.

13.14(11) says:

> At the place where a name causes freezing, the entity denoted by the name is frozen, unless the name is a prefix of an expanded name; at the place where an object name causes freezing, the nominal subtype associated with the name is frozen.

And AARM 13.14(11.a-11.b) say:

> Ramification: This only matters in the presence of deferred constants or access types; an object_declaration other than a deferred_constant_declaration causes freezing of the nominal subtype, plus all component junk.
>
> Implicit_dereferences are covered by expression.

It seems that AARM 13.14(11.b) is wrong -- an implicit_dereference is *not* an expression.

---

3. Does an implicit call to Initialize freeze the subprogram? (Yes.) The freezing rules seem to apply to explicit constructs. For example:

```
type T is new Controlled with record...;
procedure Initialize(X: in out T);
X: T; -- Implicit call to Initialize.
for Initialize'Address use ...; -- Legal?  (No.)
```

If this is legal, it will raise Program_Error, but AARM 13.14(1.o) argues that that's no excuse.

The same question applies to Adjust and Finalize, and also to implicit calls to user-defined storage pool operations.

---

4. It seems unclear whether an implicit type conversion freezes. For example:

```
type Color is (Red, Yellow);
subtype S is Color range Red..Red; -- The expression "Red" freezes type
Color.
```

But:

```
type T is range 1..100;
subtype S is T range 1..10; -- Freezes type T?  (Yes.)
```

The expressions "1" and "10" are of type universal_integer, so T is not frozen. But it seems like it should be -- the value is implicitly converted to type T, and so it's very much like an expression of type T.

13.14(12) seems to agree that the implicit conversion should freeze.

> At the place where a range causes freezing, the type of the range is frozen.

Here's a case not covered by 13.14(12):

```
type T is range 1..10;
function F(X: T) return boolean;
X: Boolean := F(10); -- Freezes type T?  (Yes.)
for T'Size use 4; -- Legal?  (No.)
```

Summary of Response

1. An object name causes freezing where it occurs, unless the name is part of a default_expression, a default_name, or a per-object expression of a component's constraint, in which case, the freezing occurs later as part of another construct.

2. An implicit_dereference freezes entities according to the same rule that applies to a name that is an explicit_dereference.

3. An implicit call, such as an implicit call to Initialize, freezes the called subprogram. This is true even if the implicit call is removed via the implementation permissions in 7.6(18-21).

4. If a name or expression is implicitly converted to a type or subtype, then that type or subtype is frozen at the same place where the name or expression causes freezing.

Corrigendum Wording

Replace 13.14(4):

> A construct that (explicitly or implicitly) references an entity can cause the *freezing* of the entity, as defined by subsequent paragraphs. At the place where a construct causes freezing, each name, expression, or range within the construct causes freezing:

by:

> A construct that (explicitly or implicitly) references an entity can cause the *freezing* of the entity, as defined by subsequent paragraphs. At the place where a construct causes freezing, each name, expression, implicit_dereference, or range within the construct causes freezing:

Replace 13.14(8):

A static expression causes freezing where it occurs. A nonstatic expression causes freezing where it occurs, unless the expression is part of a default_expression, a default_name, or a per-object expression of a component's constraint, in which case, the freezing occurs later as part of another construct.

by:

A static expression causes freezing where it occurs. An object name or nonstatic expression causes freezing where it occurs, unless the name or expression is part of a default_expression, a default_name, or a per-object expression of a component's constraint, in which case, the freezing occurs later as part of another construct.

An implicit call freezes the same entities that would be frozen by an explicit call. This is true even if the implicit call is removed via implementation permissions.

If an expression is implicitly converted to a type or subtype *T*, then at the place where the expression causes freezing, *T* is frozen.

Insert after 13.14(11):

- At the place where a name causes freezing, the entity denoted by the name is frozen, unless the name is a prefix of an expanded name; at the place where an object name causes freezing, the nominal subtype associated with the name is frozen.

the new paragraph:

- At the place where an implicit_dereference causes freezing, the nominal subtype associated with the implicit_dereference is frozen.

Discussion

1. 13.14(8) says that expressions cause freezing. It does not cover names that denote objects, but it should -- the reasons for the existence of 13.14(8) apply equally to object names.

Given the conclusion herein reached, the above example (1) is illegal. The occurrence of "X.all" freezes the type T, but the type is not completely defined at that point, thus violating 13.14(17). Note that the declaration of Y is an object_renaming_declaration, not an object_declaration, so 13.14(6) does not apply.

If the above example (1) were legal, it would necessarily raise Constraint_Error due to dereferencing a null access value. However, AARM 13.14(1.o-1.u) explains that we do not wish to rely on run-time checks for this kind of example. Furthermore, it is possible to construct examples that do not necessarily raise an exception.

Object_renaming_declarations are not the only offender. Here's another example:

```
with System.Storage_Pools; use System.Storage_Pools;
package Q is
    type My_Pool is new Root_Storage_Pool with private;
    type My_Pool_Ptr is access all My_Pool;
    Ptr: My_Pool_Ptr;

    type Acc is access Integer;
    for Acc'Storage_Pool use Ptr.all; -- Illegal!
private
    type My_Pool is new Root_Storage_Pool with ...;
end Q;
```

The above is illegal because the name "Ptr.all" freezes type My_Pool before My_Pool is completely defined.

The problem occurs in any case where an object name can occur, and is analogous to the expression case in 13.14(8); hence the resolution is worded by analogy with 13.14(8).

---

2. Clearly, the same rules should apply to explicit and implicit dereferences -- in the example, "Obj.all.D" and "Obj.D" should freeze the same entities. Therefore, a new bullet after 13.14(11) is added to cover implicit_dereferences, so that the "Obj" in "Obj.D" freezes the same entities that "Obj.all" would freeze. That is, an implicit_dereference freezes the denoted object and its nominal subtype.

Since an implicit_dereference is not an expression and is not a name (although it may be part of a name), it is added to 13.14(4).

---

3 and 4. Clearly implicit calls and implicit conversions should freeze in the same manner as their explicit counterparts. An implicit call should freeze even if it is removed via the implementation permissions in 7.6(18-21); otherwise, there would be a portability problem.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0047**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **A**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0047 Integer_Text_IO, etc. not listed in A(2)

Working Reference Number AI95-00081

Question

Integer_Text_IO and Float_Text_IO are not listed in A(2), but Elementary_Functions (for example) is listed. Is this intended? (No.)

Summary of Response

Integer_Text_IO and Float_Text_IO should be listed in A(2). Similarly, Integer_Wide_Text_IO and Float_Wide_Text_IO should be listed in A(2).

Corrigendum Wording

In A(2) replace:

Finalization — 7.6

Interrupts — C.3.2

by:

Finalization — 7.6

Float_Text_IO — A.10.9

Float_Wide_Text_IO — A.11

Integer_Text_IO — A.10.8

Integer_Wide_Text_IO — A.11

Interrupts — C.3.2

Discussion

This was an oversight.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0048**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.4.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0048 Bounds of string returned by Ada.Strings.Maps.To_Range

Working Reference Number AI95-00151

Question

A.4.2(63) says:

> To_Range returns the Character_Sequence value R, with lower bound 1 and upper bound Map'Length, such that if D = To_Domain(Map) then D(I) maps to R(I) for each I in D'Range.

However, Map is not an array, so Map'Length makes no sense.

Summary of Response

To_Range returns the Character_Sequence value R, such that if D = To_Domain (Map), then R has the same bounds as D, and D(I) maps to R(I) for each I in D'Range.

Corrigendum Wording

Replace A.4.2(63):

> To_Range returns the Character_Sequence value R, with lower bound 1 and upper bound Map'Length, such that if D = To_Domain(Map) then D(I) maps to R(I) for each I in D'Range.

by:

> To_Range returns the Character_Sequence value R, such that if D = To_Domain (Map), then R has the same bounds as D, and D(I) maps to R(I) for each I in D'Range.

Discussion

The simplest fix is to specify that the bounds are the same as those for To_Domain.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0049**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.4.3; A.4.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0049 String packages

Working Reference Number AI95-00128

Question

1. The string packages (e.g., Ada.Strings.Fixed) have a procedure named Find_Token whose profile is:

```
procedure Find_Token (Source : in String;
                      Set : in Maps.Character_Set;
                      Test : in Membership;
                      First : out Positive;
                      Last : out Natural);
```

The semantics of this operation states that (A.4.3(68)) "if no such slice exists, then the value returned for Last is zero, and the value returned for First is Source'First."

What happens when Source'First is not in Positive (which can happen only if Source is a null string)? (It raises Constraint_Error.)

2. The semantics of Bounded.Slice is stated as follows (A.4.4(101)): "Returns the slice at positions Low through High in the string represented by Source; propagates Index_Error if Low > Length(Source)+1."

What happens when Low <= Length(Source)+1 and High > Length(Source)? Should it raise an exception? If so which one? Or should it return all characters from Low to Length(Source)? (It raises Index_Error.)

3. The semantics of many subprograms of package Bounded is defined in terms of the semantics of the corresponding subprograms of package Fixed (A.4.4(102-105)). The meaning is clear in most cases, except for Head and Tail.

A.4.4(105) says: "Each of the ... selector subprograms (Trim, Head, Tail) ... has an effect based on its corresponding subprogram in Strings.Fixed ..."

The procedure Fixed.Head has the following profile:

```
procedure Head (Source : in out String;
                Count : in Natural;
                Justify : in Alignment := Left;
                Pad : in Character := Space);
```

and the procedure Bounded.Head has a rather different profile:

```
procedure Head (Source : in out Bounded_String;
                Count : in Natural;
                Pad : in Character := Space;
                Drop : in Truncation := Error);
```

Because the profiles are different, the "effect based on the corresponding subprogram" is not very clear. It is interesting to note that the semantics of the operations of package Unbounded makes a distinction between functions and procedures (A.4.5(86-87)), which clarifies very much the meaning. Is the intent similar for Bounded?

The issue seems to be broader than Head and Tail: take for instance procedure Bounded.Replace_Slice. Is it based on the function Fixed.Replace_Slice, or on the procedure Fixed.Replace_Slice? The effect is rather different, since the procedure doesn't change the length of its argument, while the function may return a string of a different length than its argument.

4. A.4.3(2) says:

> For each function that returns a String, the lower bound of the returned value is 1.

However, A.4.3(73) says:

```
function Replace_Slice (Source   : in String;
                        Low      : in Positive;
                        High     : in Natural;
                        By       : in String)
        return String;
```

> If Low > Source'Last+1, or High < Source'First-1, then Index_Error is propagated. Otherwise, if High >= Low then the returned string comprises Source(Source'First..Low-1) & By & Source(High+1..Source'Last), and if High < Low then the returned string is Insert(Source, Before=>Low, New_Item=>By).

The lower bounds of the above concatenations give Source'First as the lower bound, which might not be 1.

Is the lower bound really 1? (Yes.)

Summary of Response

This resolution clarifies minor details of the semantics of some of the string-manipulation subprograms:

1. Fixed.Find_Token raises Constraint_Error if the value returned for First is not in Positive.

2. A call to Bounded.Slice with High > Length(Source) raises Index_Error.

3. The functions in Bounded, such as Replace_Slice, are defined in terms of the corresponding functions in Fixed, and the procedures in Bounded are defined in terms of the functions in Bounded.

4. A.4.3(2) holds throughout A.4.3.

Corrigendum Wording

Replace A.4.3(68):

> Find_Token returns in First and Last the indices of the beginning and end of the first slice of Source all of whose elements satisfy the Test condition, and such that the elements (if any) immediately before and after the slice do not satisfy the Test condition. If no such slice exists, then the value returned for Last is zero, and the value returned for First is Source'First.

by:

> Find_Token returns in First and Last the indices of the beginning and end of the first slice of Source all of whose elements satisfy the Test condition, and such that the elements (if any) immediately before and after the slice do not satisfy the Test condition. If no such slice exists, then the value returned for Last is zero, and the value returned for First is Source'First; however, if Source'First is not in Positive then Constraint_Error is raised.

Replace A.4.3(74):

> If Low > Source'Last+1, or High < Source'First-1, then Index_Error is propagated. Otherwise, if High >= Low then the returned string comprises Source(Source'First..Low-1) & By & Source(High+1..Source'Last), and if High < Low then the returned string is Insert(Source, Before=>Low, New_Item=>By).

by:

> If Low > Source'Last+1, or High < Source'First-1, then Index_Error is propagated. Otherwise:
>
> - If High >= Low, then the returned string comprises Source(Source'First..Low-1) & By & Source(High+1..Source'Last), but with lower bound 1.
> - If High < Low, then the returned string is Insert(Source, Before=>Low, New_Item=>By).

Replace A.4.3(86):

If From <= Through, the returned string is Replace_Slice(Source, From, Through, ""), otherwise it is Source.

by:

If From <= Through, the returned string is Replace_Slice(Source, From, Through, ""), otherwise it is Source with lower bound 1.

Replace A.4.3(106):

These functions replicate a character or string a specified number of times. The first function returns a string whose length is Left and each of whose elements is Right. The second function returns a string whose length is Left*Right'Length and whose value is the null string if Left = 0 and is (Left-1)*Right & Right otherwise.

by:

These functions replicate a character or string a specified number of times. The first function returns a string whose length is Left and each of whose elements is Right. The second function returns a string whose length is Left*Right'Length and whose value is the null string if Left = 0 and otherwise is (Left-1)*Right & Right with lower bound 1.

Replace A.4.4(101):

Returns the slice at positions Low through High in the string represented by Source; propagates Index_Error if Low > Length(Source)+1.

by:

Returns the slice at positions Low through High in the string represented by Source; propagates Index_Error if Low > Length(Source)+1 or High > Length(Source).

Replace A.4.4(105):

Each of the transformation subprograms (Replace_Slice, Insert, Overwrite, Delete), selector subprograms (Trim, Head, Tail), and constructor functions ("*") has an effect based on its corresponding subprogram in Strings.Fixed, and Replicate is based on Fixed."*". For each of these subprograms, the corresponding fixed-length string subprogram is applied to the string represented by the Bounded_String parameter. To_Bounded_String is applied the result string, with Drop (or Error in the case of Generic_Bounded_Length."*") determining the effect when the string length exceeds Max_Length.

by:

Each of the transformation subprograms (Replace_Slice, Insert, Overwrite, Delete), selector subprograms (Trim, Head, Tail), and constructor functions ("*") has an effect based on its corresponding subprogram in Strings.Fixed, and Replicate is based on Fixed."*". In the case of a function, the corresponding fixed-length string function is applied to the string represented by the Bounded_String parameter. To_Bounded_String is applied to the result string, with Drop (or Error in the case of Generic_Bounded_Length."*") determining the effect when the string length exceeds Max_Length. In the case of a procedure, the corresponding function in Strings.Bounded.Generic_Bounded_Length is applied, with the result assigned into the Source parameter.

Discussion

1. Fixed.Find_Token raises Constraint_Error if the value returned for First is not in Positive. Bounded.Find_Token and Unbounded.Find_Token's string argument always has a lower bound of 1 (by definition), so the question does not apply to them.

2. A call to Bounded.Slice with High > Length(Source) raises Index_Error. This is analogous to the normal string slicing operation.

3. The *function* Bounded.Head is defined in terms of the function Fixed.Head; a call of the function Bounded.Head is equivalent to:

```
To_Bounded_String(Fixed.Head(To_String(Source), Count, Pad), Drop => Drop)
```

The *procedure* Bounded.Head is defined in terms of the *function* Bounded.Head; a call to the procedure Bounded.Head is equivalent to:

```
Source := Head(Source, Count, Pad, Drop);
```

Corresponding rules apply to Tail. In general, the functions in Bounded, such as Replace_Slice, are defined in terms of the corresponding functions in Fixed, and the procedures in Bounded are defined in terms of the functions in Bounded.

4. Clearly, the intent is that the lower bound should always be 1, as stated in A.4.3(2). A "friendly" reading is that A.4.3(74) is just telling us the characters of the string (it says "comprises", and not "is equivalent to"), and is not intended to define the bounds.

A.4.3(2) is therefore interpreted to hold throughout A.4.3.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0050**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **A.5.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0050 Float_Random.Value, Discrete_Random.Value

Working Reference Number AI95-00089

Question

A.5.2(40) says:

> Invoking Value with a string that is not the image of any generator state raises Constraint_Error.

Is it legal to allow some extra flexibility? (Yes.) For example, suppose the Image function returns a representation of the state as a string of hexadecimal digits, with 'A'..'F' in upper case. A string with 'a'..'f' in lower case, but which is otherwise equivalent to a valid image, is not strictly speaking "the image of any generator state". May the Value function nevertheless return a valid state for such a string, or must it raise Constraint_Error?

Summary of Response

It is a bounded error to invoke Value with a string that is not the image of any generator. If the error is detected, Constraint_Error or Program_Error is raised. Otherwise, a call to Reset with the resulting State will produce a generator such that calls to Random with this generator will produce a sequence of values of the appropriate subtype, but which might not be random in character. That is, the sequence of values might not fulfill the requirements of A.5.2(41-43).

Corrigendum Wording

Replace A.5.2(40):

> Invoking Value with a string that is not the image of any generator state raises Constraint_Error.

by:

> *Bounded (Run-Time) Errors*
>
> It is a bounded error to invoke Value with a string that is not the image of any generator state. If the error is detected, Constraint_Error or Program_Error is raised. Otherwise, a call to Reset with the resulting state will produce a generator such that calls to Random with this generator will produce a sequence of values of the appropriate subtype, but which might not be random in character. That is, the sequence of values might not fulfill the implementation requirements of this subclause.

Discussion

A.5.2(40) seems to imply that the implementation must detect strings that could not have been produced by Image. However, for some kinds of random number generators, such detection is prohibitively expensive. Therefore, we choose to make this situation a bounded error. If the given string is syntactically malformed, the implementation will probably raise an exception. However, some strings might "look right", but produce a generator state that could never come from a valid seed, and results in non-random numbers.

There is no need to make the situation erroneous -- the implementation shouldn't write to random memory locations, or take wild jumps. The worst that can happen is that a non-random sequence of numbers (for example, a sequence of zeros) will be produced.

To be portable, the programmer should ensure that every string passed to Value came originally from a call to Image.

Note that A.5.2(45) says, "The implementation ... shall document the nature of the strings that Value will accept without raising Constraint_Error."

The reason for adding Program_Error to the list of possibilities is simply that 1.1.5(8) says that every bounded error can raise Program_Error.

Note that this ruling does not allow calls to Random to raise Constraint_Error or Program_Error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0051**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.10.1; A.10.3; A.12.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0051 Text_IO.Flush should use mode 'in'

Working Reference Number AI95-00057

Question

A.10.1(21) shows the parameter of Text_IO.Flush as mode 'in out'. This makes it impossible to flush Standard_Output. Is this the intent? (No.)

Summary of Response

The mode of the parameter of Text_IO.Flush is 'in' (not 'in out'). The same is true of Streams.Stream_IO.Flush.

Corrigendum Wording

Replace A.10.1(21):

```
   -- Buffer control

      procedure Flush (File : in out File_Type);
      procedure Flush;
```

by:

```
   -- Buffer control

      procedure Flush (File : in File_Type);
      procedure Flush;
```

Replace A.10.3(20):

```
      procedure Flush (File : in out File_Type);
      procedure Flush;
```

by:

```
      procedure Flush (File : in File_Type);
      procedure Flush;
```

Replace A.12.1(25):

```
      procedure Flush (File : in out File_Type);
```

by:

```
      procedure Flush (File : in File_Type);
```

Response

The parameter mode of Text_IO.Flush and Stream_IO.Flush is 'in' (not 'in out', as shown in the International Standard).

Discussion

It is important to be able to call Flush on the Current_Output, Standard_Output, Current_Error, and Standard_Error files. However, these files are only accessible via function results or dereferencing an access-to-constant value; thus they cannot be flushed if the mode is 'in out'.

Note that Current_Output is flushed by the parameterless Flush procedure (see A.10.3(21)); thus it had better make sense to flush Current_Output.

Therefore, we make the mode of the parameter of Flush be 'in'. This is consistent with procedures like Put, which also modify the file; a level of indirection is presumed in the implementation.

For consistency, the same applies to Stream_IO.Flush (see A.12.1(25)).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0052**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.10.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0052 Error in Standard_Error definition

Working Reference Number AI95-00194

Question

In the definition of the Text_IO.Standard_Error function with result type File_Access, the standard states that the returned access value designates "the standard output file." This is just a typographical error, right? (Yes.)

Summary of Response

A.10.3(12) should refer to the standard error file, not to the standard output file.

Corrigendum Wording

Replace A.10.3(12):

> Returns the standard error file (see A.10), or an access value designating the standard output file, respectively.

by:

> Returns the standard error file (see A.10), or an access value designating the standard error file, respectively.

Discussion

The intent here is obvious; this is just an editing error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0053**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.10.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0053 Erroneous execution for closing default files

Working Reference Number AI95-00063

Question

A.10.3(23) states:

> If the Close operation is applied to a file object that is also serving as the default input, default output, or default error file, then subsequent operations on such a default file are erroneous.

This seems to imply that once the Close operation is done to, say, the default output file, any further reference to the default output file is erroneous, even if the default output file has been set to a different (open) file. That is, closing a file that happens to be the default output file poisons any reference to a *different* default output file. Is this the intent? (No.)

Consider:

```
Set_Output(File_1);
Close(File_1);
Set_Output(File_2);
Put(X); -- Erroneous?  (No.)
```

Summary of Response

An operation on a default file is erroneous if the corresponding file object is closed at the time the operation is invoked.

Corrigendum Wording

Replace A.10.3(22):

> The execution of a program is erroneous if it attempts to use a current default input, default output, or default error file that no longer exists.

by:

> The execution of a program is erroneous if it invokes an operation on a current default input, default output, or default error file, and if the corresponding file object is closed or no longer exists.

Delete A.10.3(23):

> If the Close operation is applied to a file object that is also serving as the default input, default output, or default error file, then subsequent operations on such a default file are erroneous.

Discussion

The above interpretation makes the most sense -- it doesn't make sense for a close to forever poison the default file. That is, if you close the default file, and then reference that default file, it should be erroneous, but if you close the default file, then reset the default file to refer to some other open file, then it should not be erroneous to then reference the default file.

In the above example, if there were no "Set_Output(File_2);", then execution would be erroneous.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0054**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.10.10**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0054 Enumeration_IO does not allow instantiation for a float type

Working Reference Number AI95-00007

Question

A.10.10(17) says: "Enumeration_IO would allow instantiation for an float type". This is obviously a typographical error; "integer" is meant.

Summary of Response

Enumeration_IO cannot be instantiated for a floating point type.

Corrigendum Wording

Replace A.10.10(17):

> Although the specification of the generic package Enumeration_IO would allow instantiation for an float type, this is not the intended purpose of this generic package, and the effect of such instantiations is not defined by the language.

by:

> Although the specification of the generic package Enumeration_IO would allow instantiation for an integer type, this is not the intended purpose of this generic package, and the effect of such instantiations is not defined by the language.

Response

Enumeration_IO cannot be instantiated for a floating point type.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0055**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.12.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0055 Stream_IO.Read and Stream_IO.Write advance the current index

Working Reference Number AI95-00026

Question

Which operations set and modify the current file index of a stream file?

Summary of Response

Opening a file of type Streams.Stream_IO.File_Type in mode Append_File, or resetting such a file to mode Append_File, sets the current file index to Size(File)+1. Beyond this, the current file index maintained by Stream_IO is set in the same manner as the current file index maintained by instances of Direct_IO.

Corrigendum Wording

Insert before A.12.1(2):

The library package Streams.Stream_IO has the following declaration:

the new paragraph:

The elements of a stream file are stream elements. If positioning is supported for the specified external file, a current index and current size are maintained for the file as described in A.8. If positioning is not supported, a current index is not maintained, and the current size is implementation defined.

Insert after A.12.1(28):

The subprograms Create, Open, Close, Delete, Reset, Mode, Name, Form, Is_Open, and End_of_File have the same effect as the corresponding subprograms in Sequential_IO (see A.8.2).

the new paragraphs:

The Set_Mode procedure changes the mode of the file. If the new mode is Append_File, the file is positioned to its end; otherwise, the position in the file is unchanged.

The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File.

Insert after A.12.1(30):

The procedures Read and Write are equivalent to the corresponding operations in the package Streams. Read propagates Mode_Error if the mode of File is not In_File. Write propagates Mode_Error if the mode of File is not Out_File or Append_File. The Read procedure with a Positive_Count parameter starts reading at the specified index. The Write procedure with a Positive_Count parameter starts writing at the specified index.

the new paragraph:

The Size function returns the current size of the file.

Replace A.12.1(31):

The Index function returns the current file index, as a count (in stream elements) from the beginning of the file. The position of the first element in the file is 1.

by:

The Index function returns the current index.

Insert after A.12.1(32):

The Set_Index procedure sets the current index to the specified value.

the new paragraphs:

If positioning is supported for the external file, the current index is maintained as follows:

- For Open and Create, if the Mode parameter is Append_File, the current index is set to the current size of the file plus one; otherwise, the current index is set to one.
- For Reset, if the Mode parameter is Append_File, or no Mode parameter is given and the current mode is Append_File, the current index is set to the current size of the file plus one; otherwise, the current index is set to one.
- For Set_Mode, if the new mode is Append_File, the current index is set to current size plus one; otherwise, the current index is unchanged.
- For Read and Write without a Positive_Count parameter, the current index is incremented by the number of stream elements read or written.
- For Read and Write with a Positive_Count parameter, the value of the current index is set to the value of the Positive_Count parameter plus the number of stream elements read or written.

Delete A.12.1(34):

The Size function returns the current size of the file, in stream elements.

Delete A.12.1(35):

The Set_Mode procedure changes the mode of the file. If the new mode is Append_File, the file is positioned to its end; otherwise, the position in the file is unchanged.

Delete A.12.1(36):

The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File.

Response

The following operations set the value of the current index if positioning is supported for the specified file:

- Open(File,Mode,Name,Form) and Create(File,Mode,Name,Form) set the current index to Size(File)+1 if Mode(File) = Append_File, and to 1 otherwise.

- Read increments the current index by the number of stream elements read.

- Write increments the current index by the number of stream elements written.

- Set_Index(File,To) sets the current index to the value of To (which may be greater than Size(File)).

- Set_Mode(File,Mode) sets the current index to Size(File)+1 if Mode = Append_File, and leaves it unchanged otherwise.

- Reset(File,Mode) sets the current index to Size(File)+1 if Mode = Append_File, and to 1 otherwise; Reset(File) sets the current index to Size(File)+1 if Mode(File) = Append_File, and to 1 otherwise.

Set_Index and the versions of Read and Write with Positive_Count parameters raise Use_Error if positioning is not supported for the specified file.

Discussion

A.12.1 describes the current index, or position, of a stream file, but does not indicate that its value is set by any operation other than Set_Index and Set_Mode. Stream files also fail to describe their conceptual model.

The intent was that stream files that support positioning are similar to direct files, and that other stream files are similar to sequential files. Both of these file types are described in A.8. Note, however, that A.8 specifically says it does not apply to stream files.

Since stream files with positioning are intended to be similar to direct files, the current index of a stream file should be handled similarly (except that the index counts stream elements rather than file elements, and except that a stream file can be opened in or reset to mode Append_File). The recommendation is based on the behavior described in A.8(4), A.8.2, and A.8.5 for direct files.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0056**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **A.12.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0056 Ada.Streams.Stream_IO.Stream can raise Status_Error

Working Reference Number AI95-00001

Question

If the function Ada.Streams.Stream_IO.Stream is called with a closed file, does the call raise Status_Error? (Yes.)

If the function call does not raise Status_Error, does an attempt to read from or write to the stream referenced by the resulting access value raise Status_Error if the file has been closed? (It is erroneous.)

Summary of Response

Ada.Streams.Stream_IO.Stream raises Status_Error if its parameter is not an open file. If the file passed to the Stream function is closed or ceases to exist after the call on the Stream function and the Root_Stream_Type'Class object designated by the function result is subsequently passed as the first parameter to Ada.Streams.Read or Ada.Streams.Write, execution is erroneous.

Corrigendum Wording

Replace A.12.1(29):

> The Stream function returns a Stream_Access result from a File_Type object, thus allowing the stream-oriented attributes Read, Write, Input, and Output to be used on the same file for multiple types.

by:

> The Stream function returns a Stream_Access result from a File_Type object, thus allowing the stream-oriented attributes Read, Write, Input, and Output to be used on the same file for multiple types. Stream propagates Status_Error if File is not open.

Insert after A.12.1(36):

> The Flush procedure synchronizes the external file with the internal file (by flushing any internal buffers) without closing the file or changing the position. Mode_Error is propagated if the mode of the file is In_File.

the new paragraph:

> *Erroneous Execution*
>
> If the File_Type object passed to the Stream function is later closed or finalized, and the stream-oriented attributes are subsequently called (explicitly or implicitly) on the Stream_Access value returned by Stream, execution is erroneous. This rule applies even if the File_Type object was opened again after it had been closed.

Response

The result of Ada.Streams.Stream_IO.Stream is associated with a specific opening of the file passed as a parameter.

A call on Stream raises Status_Error if its parameter is not an open file.

If the file passed to Stream is closed after the call on Stream and the Root_Stream_Type'Class object designated by the function result is subsequently passed as the first parameter to Ada.Streams.Read or Ada.Streams.Write, execution is erroneous, even if the file was opened again after it was closed. (Such calls on Read or Write may arise indirectly from calls on the subprograms denoted by the stream-oriented attributes.) Similarly, if the file passed to Stream ceases to exist after the call on Stream (e.g., upon exit from the scope in which the File_Type object was declared) and the object designated by the function result is subsequently passed to Read or Write, execution is erroneous.

It follows from A.12.2(5) that the same rules apply to Ada.Text_IO.Text_Streams.Stream. It follows from A.12.3(5) that the same rules apply to Ada.Wide_Text_IO.Text_Streams.Stream .

Discussion

Open, Create, and Is_Open are the only subprograms in predefined program units that can be invoked with a closed file without raising Status_Error.

The rules stipulating when use of the result of the Stream function is erroneous are analogous to the rules in A.10.3(22) and A.10.3(23) for the result of the Current_Input, Current_Output, and Current_Error functions. These rules make it possible to represent a File_Type value or a file stream-value as an access value, with a null value corresponding to a closed file. (By a file-stream value, we mean a value belonging to some descendant of Root_Stream_Type and representing a stream associated with a file.)

The risk of erroneous execution can be minimized by using the Stream function only as an actual parameter to Ada.Streams.Read, Ada.Streams.Write, or a stream-oriented attribute.

An alternative approach would be to allow the result of the Stream function to correspond to a closed file, but to raise Status_Error upon an attempt to use a file-stream value associated with a closed or no longer existent file. Then a file-stream value would have to reflect the fact that its corresponding internal file had been closed or had ceased to exist. This would rule out an implementation in which closing a file simply sets a File_Type value to a null pointer. Finalization of both File_Type objects and file-stream objects would be complicated. (One possible implementation would be for both File_Type objects and file-stream objects to be controlled objects pointing to an object that includes both a reference count and an is-opened flag. Such an object would be allocated upon creation of a File_Type object, but would continue to exist beyond the lifetime of the File_Type object if there were still unfinalized file-stream objects pointing to it.)

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0057**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **A.14**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0057 External files of Standard_Input and Standard_Output

Working Reference Number AI95-00050

Question

A.14(2-3) say:

> Operations on one text file object do not affect the column, line, and page numbers of any other file object.
>
> Standard_Input and Standard_Output are associated with distinct external files, so operations on one of these files cannot affect operations on the other file. In particular, reading from Standard_Input does not affect the current page, line, and column numbers for Standard_Output, nor does writing to Standard_Output affect the current page, line, and column numbers for Standard_Input.

What is the intended meaning of the statement, "Standard_Input and Standard_Output are associated with distinct external files", given that the operating system may well consider standard input and standard output to be associated with the same device (say, a terminal)?

The NOTE in A.10.3(25) contradicts A.14(3):

> 24 The standard input, standard output, and standard error files are different file objects, but not necessarily different external files.

A.10(5-6) also discuss these files:

> At the beginning of program execution the default input and output files are the so-called standard input file and standard output file. These files are open, have respectively the current modes In_File and Out_File, and are associated with two implementation-defined external files. Procedures are provided to change the current default input file and the current default output file.
>
> At the beginning of program execution a default file for program-dependent error-related text output is the so-called standard error file. This file is open, has the current mode Out_File, and is associated with an implementation-defined external file. A procedure is provided to change the current default error file.

Summary of Response

Standard_Input, Standard_Output, and Standard_Error are associated with three distinct (internal) file objects. Their association with external files is not specified by the language; in particular, these three external files need not be distinct.

Corrigendum Wording

Delete A.14(3):

- Standard_Input and Standard_Output are associated with distinct external files, so operations on one of these files cannot affect operations on the other file. In particular, reading from Standard_Input does not affect the current page, line, and column numbers for Standard_Output, nor does writing to Standard_Output affect the current page, line, and column numbers for Standard_Input.

Discussion

The NOTE in A.10.3(25) is correct. A.10(5-6) do not specifically say whether the external files are distinct. In any case, external files are "external" from the point of view of Ada semantics, so it's hard to see how one

could write a test case that distinguishes whether the referenced statement is obeyed by an alleged implementation.

On most operating systems, it is possible for Standard_Input and Standard_Output to be associated with the same external file. This happens by default, when they are both associated with the same terminal device. It can also happen when the user redirects I/O to the same file.

It is not clear that A.14(3) is trying to say anything in addition to what A.14(2) already says for all text files. Therefore, A.14(3) should simply be removed.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0058**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **B.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0058 What are the rules for named notation in pragmas?

Working Reference Number AI95-00036

Question

The syntax given for pragma Import is

```
pragma Import (
   [Convention => ] convention_identifier        -- (1)
   , [Entity =>] local_name                      -- (2)
   [ , [External_Name => ] string_expression ]   -- (3)
   [ , [Link_Name => ] string_expression ] ) ;   -- (4)
```

(and similarly for pragma Export).

Does this imply that named notation cannot be used to reorder the arguments? For example, is the following pragma legal? (No.)

```
pragma Import(C, Raise_Signal,
   Link_Name => "raise", External_Name => "._raise"); -- Illegal!
```

Secondly, is there a rule from which I can deduce that

```
pragma Import (C, Raise_Signal, "raise");
```

means

```
pragma Import (C, Raise_Signal, External_Name => "raise");
```

(by eliding the text in the outer brackets on line (4) and the text in the inner brackets on line (3)) rather than

```
pragma Import (C, Raise_Signal, Link_Name => "raise");
```

(by eliding the text in the outer brackets on line (3) and the text in the inner brackets on line(4))?

Summary of Response

A pragma must obey the syntax rules for that particular pragma. In particular, arguments written in named notation must not be given in a different order than is required by the syntax rules.

For pragma arguments written in positional notation, the first argument corresponds to the first argument shown in the syntax rule for the pragma, the second argument corresponds to the second, and so on. This is true even in the presence of optional arguments.

Corrigendum Wording

Insert after B.1(9):

> A pragma Linker_Options is allowed only at the place of a declarative_item.

the new paragraph:

> For pragmas Import and Export, the argument for Link_Name shall not be given without the pragma_argument_identifier unless the argument for External_Name is given.

Discussion

As mentioned in AARM 2.8(11.i), it is not the intent to allow reordering, even when named notation is used.

For positional notation, the intent is that pragma arguments follow the same rules as subprograms -- if positional notation is used, the arguments are associated in order. Thus,

```
   pragma Import (C, Raise_Signal, "raise");
```

means:

```
   pragma Import (C, Raise_Signal, External_Name => "raise");
```

since External_Name is the third argument of pragma Import.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0059**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **B.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0059 Interface to C — passing records as parameters of mode 'in'

Working Reference Number AI95-00131

Question

The implementation advice B.3(69) says:

> An Ada parameter of a record type T, of any mode, is passed as a t* argument to a C function, where t is the C struct corresponding to the Ada type T.

The problem with this is that if one has a C function that is passed a struct, then how can one pass an Ada record to that? One might think that if the Ada record is passed as an 'in' parameter, it will work. However, the above implementation advice implies that such an 'in' parameter will correspond to a t* on the C side, rather than a t.

Summary of Response

The implementation advice in B.3(69) is correct as written.

An implementation which supports interfacing to C shall support pragma Convention with a C_Pass_By_Copy identifier. An 'in' parameter of a C_Pass_By_Copy-compatible type T should be passed as a t argument to a C function, where t is the C struct corresponding to type T.

Corrigendum Wording

Replace B.3(1):

The facilities relevant to interfacing with the C language are the package Interfaces.C and its children; and support for the Import, Export, and Convention pragmas with *convention*_identifier C.

by:

The facilities relevant to interfacing with the C language are the package Interfaces.C and its children; support for the Import, Export, and Convention pragmas with *convention*_identifier C; and support for the Convention pragma with *convention*_identifier C_Pass_By_Copy.

Insert after B.3(60):

> The To_C and To_Ada subprograms that convert between Wide_String and wchar_array have analogous effects to the To_C and To_Ada subprograms that convert between String and char_array, except that wide_nul is used instead of nul.

the new paragraphs:

A Convention pragma with *convention*_identifier C_Pass_By_Copy shall only be applied to a type.

The eligibility rules in B.1 do not apply to convention C_Pass_By_Copy. Instead, a type T is eligible for convention C_Pass_By_Copy if T is a record type that has no discriminants and that only has components with statically constrained subtypes, and each component is C-compatible.

If a type is C_Pass_By_Copy-compatible then it is also C-compatible.

Replace B.3(61):

An implementation shall support pragma Convention with a C *convention*_identifier for a C-eligible type (see B.1)

by:

An implementation shall support pragma Convention with a C *convention*_identifier for a C-eligible type (see B.1). An implementation shall support pragma Convention with a C_Pass_By_Copy *convention*_identifier for a C_Pass_By_Copy-eligible type.

Insert after B.3(68):

- An Ada **access** T parameter, or an Ada **out** or **in out** parameter of an elementary type T, is passed as a t* argument to a C function, where t is the C type corresponding to the Ada type T. In the case of an elementary **out** or **in out** parameter, a pointer to a temporary copy is used to preserve by-copy semantics.

the new paragraph:

- An Ada parameter of a C_Pass_By_Copy-compatible (record) type T, of mode **in**, is passed as a t argument to a C function, where t is the C struct corresponding to the Ada type T.

Replace B.3(69):

- An Ada parameter of a record type T, of any mode, is passed as a t* argument to a C function, where t is the C struct corresponding to the Ada type T.

by:

- An Ada parameter of a record type T, of any mode, other than an **in** parameter of a C_Pass_By_Copy-compatible type, is passed as a t* argument to a C function, where t is the C struct corresponding to the Ada type T.

Response

The implementation advice in B.3(69) is left unchanged (that is, C-compatible records are passed by reference).

The convention C_Pass_By_Copy is added to the facilities available for interfacing with C. This convention can only be used in pragma Convention (not in pragmas Import or Export) and only when this pragma is applied to a type.

There is no language interface package corresponding to C_Pass_By_Copy. In other words, B.1(13) never applies to convention C_Pass_By_Copy, and there is no package named Interfaces.C_Pass_By_Copy.

A type T is eligible for convention C_Pass_By_Copy if T is a record type that has no discriminants and that only has components with statically constrained subtypes, and each component is C-compatible. (The eligibility rules in B.1(13-18) do not apply to convention C_Pass_By_Copy.)

If a type is C_Pass_By_Copy-compatible then it is also C-compatible.

An implementation supporting interfacing to C shall support pragma Convention with a C_Pass_By_Copy convention_identifier for a C_Pass_By_Copy-eligible type.

The following sentence is added to the implementation advice in B.3(64-71):

An Ada parameter of a C_Pass_By_Copy-compatible (record) type T, of mode in, should be passed as a t argument to a C function, where t is the C struct corresponding to the Ada type T.

Note that the rules B.1(19) and B.1(20) apply to convention C_Pass_By_Copy. In particular, an implementation may permit other types as C_Pass_By_Copy-compatible types (e.g., discriminated records).

Discussion

It was a mistake to require pass-by-reference for records passed to C functions. However, at this point, it would be disruptive to change the rule, and there is an alternative (see below).

The most important use of this interface is to take an existing C interface, and use it from Ada code (as opposed to taking an existing Ada interface, and mapping it to some corresponding C code).

Structs are passed by copy in C. This can be implemented by passing a copy of the struct (on the stack, in a register, or whatever), or by making a copy at the call site, and passing the address of that copy. Either way,

whatever the C compiler does, the goal should be for the Ada compiler to mimic the C compiler's method of passing structs (not pointers to structs).

Nonetheless, we choose to keep the implementation advice as is. Instead, we solve the problem by defining a new convention, C_Pass_By_Copy:

```
pragma Convention (C_Pass_By_Copy, T);
```

The effect is that any 'in' parameter of the type T is passed by copy to a subprogram of convention C, i.e., in a manner consistent with what C expects if the corresponding formal in the C prototype is a struct (rather than a pointer to a struct).

In order to make sure that this solution is portable, an implementation that supports interfacing to C is required to support convention C_Pass_By_Copy.

Note that there is no issue for modes 'in out' and 'out'; C doesn't have these modes, and the closest correspondence to C is a pointer-to-struct argument.

Although this is not explicitly stated in the International Standard, it is clear that an Ada function with result type T corresponds to a C function with return type t, where t is the C type corresponding to the Ada type T.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0060**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **B.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0060 In Interfaces.C, nul and wide_nul represent zero

Working Reference Number AI95-00037

Question

The following declarations appear in Interfaces.C (B.3):

```
(19)   type char is <implementation-defined character type>;

(20)   nul : constant char := char'First;
...
(30)   type wchar_t is <implementation-defined>;

(31)   wide_nul : constant wchar_t := wchar_t'First;
```

The declaration of wide_nul seems to imply that wchar_t supports the attribute First. What is the intent?

If char and/or wchar_t are signed integer types in the interfaced C implementation, may the Ada implementation reflect that fact by using a signed representation for char and/or wchar_t? (Yes.)

Note that if char and wchar_t have a signed representation, then char'First and wchar_t'First will not have a zero representation. Are the constants nul and wide_nul intended to be represented as zero? (Yes.)

Summary of Response

In package Interfaces.C, the type wchar_t is a discrete type. The constants nul and wide_nul have implementation-defined values, which should have a representation of zero. Types char and wchar_t may use a signed or unsigned representation.

Corrigendum Wording

Replace B.3(20):

```
nul : constant char := char'First;
```

by:

```
nul : constant char := implementation-defined;
```

Replace B.3(30):

```
type wchar_t is implementation-defined;
```

by:

```
type wchar_t is <implementation-defined discrete type>;
```

Replace B.3(31):

```
wide_nul : constant wchar_t := wchar_t'First;
```

by:

```
wide_nul : constant wchar_t := implementation-defined;
```

Insert before B.3(63):

An implementation should support the following interface correspondences between Ada and C.

the new paragraph:

The constants nul and wide_nul should have a representation of zero.

Discussion

The intent is that wchar_t be discrete.

The type char may have a signed representation. For example, the implementation might have:

```
for char use (-128, -127, ..., 127);
```

In that case, char'First is the wrong value to use for nul; the intent is that nul be represented as zero.

Similarly, wchar_t could be an enumeration type with a signed representation, as for char. Wchar_t could also be a signed integer type. Either way, wchar_t'First is the wrong value to use for wide_nul.

It is important to allow signed representations of char and wchar_t, in order to properly match what the C implementation does.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0061**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **B.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0061 Semantics of Interfaces.C.Strings.To_Char_Ptr when Nul_Check is False

Working Reference Number AI95-00140

Question

B.3.1(23-24) say:

```
function To_Chars_Ptr (Item      : in char_array_access;
                       Nul_Check : in Boolean := False)
    return chars_ptr;
```

> If Item is null, then To_Chars_Ptr returns Null_Ptr. Otherwise, if Nul_Check is True and Item.all does not contain nul, then the function propagates Terminator_Error; if Nul_Check is True and Item.all does contain nul, To_Chars_Ptr performs a pointer conversion with no allocation of memory.

This does not seem to cover the case where Nul_Check is False.

Summary of Response

If Nul_Check is False, Interfaces.C.Strings.To_Char_Ptr performs a pointer conversion with no allocation of memory.

Corrigendum Wording

Replace B.3.1(24):

> If Item is **null**, then To_Chars_Ptr returns Null_Ptr. Otherwise, if Nul_Check is True and Item.**all** does not contain nul, then the function propagates Terminator_Error; if Nul_Check is True and Item.**all** does contain nul, To_Chars_Ptr performs a pointer conversion with no allocation of memory.

by:

> If Item is **null**, then To_Chars_Ptr returns Null_Ptr. If Item is not **null**, Nul_Check is True, and Item.**all** does not contain nul, then the function propagates Terminator_Error; otherwise To_Chars_Ptr performs a pointer conversion without allocation of memory.

Discussion

This obvious omission is hereby corrected (see Summary of Response).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0062**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **B.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0062 Interfaces.C.Strings.Value raises Constraint_Error when Length is 0

Working Reference Number AI95-00139

Question

B.3.1(36) says, "The lower bound of the result is 0." What happens if the Length is also 0, so that there is no possible upper bound?

Summary of Response

A call to Interfaces.C.Strings.Value with Length 0, as in:

```
Value(Item => X, Length => 0)
```

raises Constraint_Error.

Corrigendum Wording

Replace B.3.1(36):

> If Item = Null_Ptr then Value(Item) propagates Dereference_Error. Otherwise Value returns the shorter of two arrays: the first Length chars pointed to by Item, and Value(Item). The lower bound of the result is 0.

by:

> If Item = Null_Ptr, then Value propagates Dereference_Error. Otherwise, Value returns the shorter of two arrays, either the first Length chars pointed to by Item, or Value(Item). The lower bound of the result is 0. If Length is 0, then Value propagates Constraint_Error.

Response

Any attempt to create a null array of type char_array, whose lower bound is 0, will clearly raise Constraint_Error. Therefore, "Value(Item => X, Length => 0)" will raise Constraint_Error. (The standard should have made this more explicit, however.)

Note that this is not harmful, since type char_array is supposed to represent a nul-terminated string, and so should not normally be of zero length.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0063**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **B.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0063 Interfaces.C.Strings.Value with Length returning String

Working Reference Number AI95-00177

Question

The definition of the function Interfaces.C.Strings.Value which takes a chars_ptr and a length, and returning a String, seems wrong. As defined, it raises Terminator_Error anytime the null character is not found before hitting the specified length. This is because of the definition of To_Ada with Trim_Null => True.

Validation test cxb3011 in suite 2.1 seems to presume that no Terminator_Error should be raised when the input chars_ptr does not have a null within the specified length.

What is the intent?

Summary of Response

A call to the following function declared in Interfaces.C.Strings:

```
function Value (Item : in chars_ptr; Length : in size_t)
   return String;
```

is equivalent to:

```
To_Ada( Value(Item, Length) & nul, Trim_Nul => True)
```

Corrigendum Wording

Replace B.3.1(40):

Equivalent to To_Ada(Value(Item, Length), Trim_Nul=>True).

by:

Equivalent to To_Ada(Value(Item, Length) & nul, Trim_Nul => True).

Discussion

B.3.1(40) says:

> Equivalent to To_Ada(Value(Item, Length), Trim_Nul=>True).

However, this is incorrect. It makes no sense to trim the nul by default, and then complain about a missing nul.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0064**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **B.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0064 Effect of Update(Null_Ptr,...)

Working Reference Number AI95-00039

Question

Does Update raise Dereference_Error if Item = Null_Ptr? (Yes.)

Summary of Response

Interfaces.C.Update raises Dereference_Error if Item = Null_Ptr.

Corrigendum Wording

Replace B.3.1(44):

> This procedure updates the value pointed to by Item, starting at position Offset, using Chars as the data to be copied into the array. Overwriting the nul terminator, and skipping with the Offset past the nul terminator, are both prevented if Check is True, as follows:

by:

> If Item = Null_Ptr, then Update propagates Dereference_Error. Otherwise, this procedure updates the value pointed to by Item, starting at position Offset, using Chars as the data to be copied into the array. Overwriting the nul terminator, and skipping with the Offset past the nul terminator, are both prevented if Check is True, as follows:

Response

Interfaces.C.Update raises Dereference_Error if Item = Null_Ptr.

Discussion

This seems like the only sensible semantics.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0065**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **B.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0065 Incorrect example for Interfaces.C.Pointers

Working Reference Number AI95-00142

Question

In the example, the usage of "=" in:

```
exit when Element = C.Nul;
```

is illegal because "=" is not visible for type C.Char.

Summary of Response

The example in B.3.2(49) should be corrected to

```
exit when C."="(Element, C.nul);
```

Corrigendum Wording

Replace B.3.2(49):

```
            loop
               Element             := Source_Temp_Ptr.all;
               Target_Temp_Ptr.all := Element;
               exit when Element = C.nul;
               Char_Ptrs.Increment(Target_Temp_Ptr);
               Char_Ptrs.Increment(Source_Temp_Ptr);
            end loop;
         end Strcpy;
      begin
         ...
      end Test_Pointers;
```

by:

```
            loop
               Element             := Source_Temp_Ptr.all;
               Target_Temp_Ptr.all := Element;
               exit when C."="(Element, C.nul);
               Char_Ptrs.Increment(Target_Temp_Ptr);
               Char_Ptrs.Increment(Source_Temp_Ptr);
            end loop;
         end Strcpy;
      begin
         ...
      end Test_Pointers;
```

Discussion

The example is wrong and should be corrected.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0066**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **B.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0066 Correction to the Valid function in COBOL interface

Working Reference Number AI95-00071

Question

The semantics of the Valid function are incorrectly stated.

Summary of Response

The Valid function should return False if Item contains leading space characters, when Format is Unsigned, Leading_Separate, or Trailing_Separate.

Corrigendum Wording

Replace B.4(63):

- Format=Unsigned: if Item comprises zero or more leading space characters followed by one or more decimal digit characters then Valid returns True, else it returns False.

by:

- Format=Unsigned: if Item comprises one or more decimal digit characters then Valid returns True, else it returns False.

Replace B.4(64):

- Format=Leading_Separate: if Item comprises zero or more leading space characters, followed by a single occurrence of the plus or minus sign character, and then one or more decimal digit characters, then Valid returns True, else it returns False.

by:

- Format=Leading_Separate: if Item comprises a single occurrence of the plus or minus sign character, and then one or more decimal digit characters, then Valid returns True, else it returns False.

Replace B.4(65):

- Format=Trailing_Separate: if Item comprises zero or more leading space characters, followed by one or more decimal digit characters and finally a plus or minus sign character, then Valid returns True, else it returns False.

by:

- Format=Trailing_Separate: if Item comprises one or more decimal digit characters followed by a plus or minus sign character, then Valid returns True, else it returns False.

Discussion

This is necessary to match COBOL.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0067**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **B.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0067 Clarification of result length for conversions in COBOL interface

Working Reference Number AI95-00072

Question

In Decimal_Conversions, the functions To_Display (B.4(71)), To_Packed (B.4(79)), and To_Binary (B.4(87)), do not specify the bounds of the result. What are these bounds?

Summary of Response

In Decimal_Conversions, the length of the result of To_Display (B.4(71)), To_Packed (B.4(79)), and To_Binary (B.4(87)) is Length(Format), and the lower bound is 1.

Corrigendum Wording

Replace B.4(71):

> This function returns the Numeric value for Item, represented in accordance with Format. Conversion_Error is propagated if Num is negative and Format is Unsigned.

by:

> This function returns the Numeric value for Item, represented in accordance with Format. The length of the returned value is Length(Format), and the lower bound is 1. Conversion_Error is propagated if Item is negative and Format is Unsigned.

Replace B.4(79):

> This function returns the Packed_Decimal value for Item, represented in accordance with Format. Conversion_Error is propagated if Num is negative and Format is Packed_Unsigned.

by:

> This function returns the Packed_Decimal value for Item, represented in accordance with Format. The length of the returned value is Length(Format), and the lower bound is 1. Conversion_Error is propagated if Item is negative and Format is Packed_Unsigned.

Replace B.4(87):

> This function returns the Byte_Array value for Item, represented in accordance with Format.

by:

> This function returns the Byte_Array value for Item, represented in accordance with Format. The length of the returned value is Length(Format), and the lower bound is 1.

Discussion

Note that the Length function is overloaded; the Format parameter to To_Display, To_Packed, or To_Binary resolves which version to use.

The description of To_Display and To_Packed contains another error: the text refers to the "value of Num", but Num is a type, not an object. The intended reference is to the parameter Item.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0068**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **C.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0068 Pragma Attach_Handler on nested objects

Working Reference Number AI95-00121

Question

C.3.1(7-8) say:

> The Attach_Handler pragma is only allowed immediately within the protected_definition where the corresponding subprogram is declared. The corresponding protected_type_declaration or single_protected_declaration shall be a library level declaration.
>
> The Interrupt_Handler pragma is only allowed immediately within a protected_definition. The corresponding protected_type_declaration shall be a library level declaration. In addition, any object_declaration of such a type shall be a library level declaration.

The AARM C.3.1(7.a) adds:

> Discussion: In the case of a protected_type_declaration, an object_declaration of an object of that type need not be at library level.

Thus, nested objects are not allowed in the Interrupt_Handler case, but they are allowed in the Attach_Handler case.

C.3.1(12) says:

> When a protected object is finalized, for any of its procedures that are attached to interrupts, the handler is detached. If the handler was attached by a procedure in the Interrupts package or if no user handler was previously attached to the interrupt, the default treatment is restored. Otherwise, that is, if an Attach_Handler pragma was used, the previous handler is restored.

and the AARM C.3.1(12.a) adds:

> Discussion: Since only library-level protected procedures can be attached as handlers using the Interrupts package, the finalization discussed above occurs only as part of the finalization of all library-level packages in a partition.

Thus, in the Attach_Handler case, when the object is finalized, the "previous handler" is restored.

What is meant by "previous handler" here? Does this feature make sense in a multi-tasking situation?

Summary of Response

A program execution is erroneous if the handlers for a given interrupt attached via pragma Attach_Handler are not attached and detached in a stack-like (LIFO) order. In particular, when a protected object is finalized, if any of its procedures are attached to interrupts via pragma Attach_Handler, then if the most recently attached handler for the same interrupt is not the same as the one that was attached at the time the protected object was created, then execution is erroneous.

Corrigendum Wording

Replace C.3.1(12):

> When a protected object is finalized, for any of its procedures that are attached to interrupts, the handler is detached. If the handler was attached by a procedure in the Interrupts package or if no user handler was previously attached to the interrupt, the default treatment is restored. Otherwise, that is, if an Attach_Handler pragma was used, the previous handler is restored.

by:

When a protected object is finalized, for any of its procedures that are attached to interrupts, the handler is detached. If the handler was attached by a procedure in the Interrupts package or if no user handler was previously attached to the interrupt, the default treatment is restored. If an Attach_Handler pragma was used and the most recently attached handler for the same interrupt is the same as the one that was attached at the time the protected object was initialized, the previous handler is restored.

Insert after C.3.1(14):

If the Ceiling_Locking policy (see D.3) is in effect and an interrupt is delivered to a handler, and the interrupt hardware priority is higher than the ceiling priority of the corresponding protected object, the execution of the program is erroneous.

the new paragraph:

If the handlers for a given interrupt attached via pragma Attach_Handler are not attached and detached in a stack-like (LIFO) order, program execution is erroneous. In particular, when a protected object is finalized, the execution is erroneous if any of the procedures of the protected object are attached to interrupts via pragma Attach_Handler and the most recently attached handler for the same interrupt is not the same as the one that was attached at the time the protected object was initialized.

Discussion

The notion of restoring the "previous handler" only makes sense if objects are created and destroyed in a stack-like (LIFO) manner. In a multi-tasking program, it is possible to do otherwise -- for example, task A declares an object, then task B declares an object, then task A completes, destroying the first object, then task B completes, destroying the second object.

Several options exist:

Option 1: require every protected object with an Attach_Handler pragma to be at library level. This is clearly not what the standard says. It doesn't completely solve the problem, either -- one could create two objects on the heap, and call an instance of Unchecked_Deallocation on them in a non-LIFO order.

Option 2: define "previous handler" to be "the handler that was attached at the time the protected object was initialized". If that handler no longer exists, execution becomes erroneous. This means that if the programmer uses a LIFO order, it all works. If the programmer uses a non-LIFO order, handlers may get restored in a "surprising" order, and in *some* cases, erroneous execution will result.

Note that it is possible to have a LIFO order, even in a multi-tasking program. For example, first declare an object at library level. Create lots of tasks. Then, at some point, one of the tasks declares another object. Clearly, this second object will be finalized before the first one, which is what we want.

The implementation in this case is not so hard: store a pointer to the previous handler in the protected object, and blindly restore it on finalization.

Option 3: define the "previous handler" to be the one that was attached just before the current handler was attached. Again, execution is erroneous if one tries to restore a handler that no longer exists. Again, the implementation is not so hard: keep a stack of handlers. When a protected object is finalized, blindly pop one item off the stack, whether or not the protected object on the stack corresponds to the current handler.

If the programmer ensures a LIFO order, then the second and third possibilities are equivalent.

Option 4: an exception is raised if a LIFO order is not obeyed. That is, when a protected object is finalized, a check is made that this protected object corresponds to the currently-attached handler; if not, an exception is raised. In this case, the implementation can be as for the second *or* the third possibility, since they are equivalent.

Option 5: same as the fourth, except that execution becomes erroneous instead of raising an exception. The implementation is the same as for the fourth possibility, except that the check is omitted.

We choose Option 5. It is the programmer's responsibility to maintain LIFO order; otherwise execution is erroneous. We do not wish to impose overhead on implementations to check for LIFO order. If an implementation wishes to check, it can raise an exception as soon as LIFO order is disobeyed (thus implementing Option 4). We also do not wish to be restrictive, as would happen with Option 1.

Upon finalization, an implementation may restore either the handler that was installed at the time the current object was initialized, or the handler that was most recently installed before the current one. For all non-erroneous situations, these two are the same handler.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0069**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **C.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0069 Parameterless_Handler values designating default treatment

Working Reference Number AI95-00166

Question

C.3.2(18) says, "The Exchange_Handler procedure operates in the same manner as Attach_Handler with the addition that the value returned in Old_Handler designates the previous treatment for the specified interrupt." This would seem to suggest that if Exchange_Handler is invoked while the default treatment is still in force, the value in Old_Handler can be dereferenced, with the dereference denoting a parameterless protected procedure that can be called to obtain the default treatment. Is this the intent? (No.)

Summary of Response

When a default treatment is in effect for an interrupt, the value returned by Current_Handler is null. Likewise, the value returned in Old_Handler by the Exchange_Handler procedure is null if it is invoked while the default treatment is in force for the specified interrupt.

Furthermore, the value returned by Current_Handler and in Old_Handler must be null whenever the treatment is not a user-defined handler.

Corrigendum Wording

Replace C.3.2(16):

> The Current_Handler function returns a value that represents the attached handler of the interrupt. If no user-defined handler is attached to the interrupt, Current_Handler returns a value that designates the default treatment; calling Attach_Handler or Exchange_Handler with this value restores the default treatment.

by:

> The Current_Handler function returns a value that represents the attached handler of the interrupt. If no user-defined handler is attached to the interrupt, Current_Handler returns **null**.

Replace C.3.2(18):

> The Exchange_Handler procedure operates in the same manner as Attach_Handler with the addition that the value returned in Old_Handler designates the previous treatment for the specified interrupt.

by:

> The Exchange_Handler procedure operates in the same manner as Attach_Handler with the addition that the value returned in Old_Handler designates the previous treatment for the specified interrupt. If the previous treatment is not a user-defined handler, **null** is returned.

Discussion

A key fact here is that a "treatment" of an interrupt (whether default or user-defined) is not limited to execution of a handler, and even if the treatment is to call a handler the handler may be something other than a parameterless protected procedure.

There may be external mechanisms (perhaps in hardware or in an operating system) that can mediate the delivery of a signal in a way that is distinct from executing a handler procedure. For example, in the UNIX environment the concept of interrupt corresponds to that of a "signal"; the default treatments for signals include ignoring (discarding) the signal or performing job-control functions including terminating the process (with or without memory dump), stopping the process, and continuing a stopped process. Even with hardware interrupts, the default treatment that is initially in place when a program starts up is unlikely to be calling an Ada protected procedure.

The intent of the International Standard is to allow treatments other than parameterless protected procedures as defaults is expressed clearly in C.3(5):

> Each interrupt has a default treatment which determines the system's response to an occurrence of that interrupt when no user-defined handler is attached. The set of possible default treatments is implementation defined, as is the method (if one exists) for configuring the default treatments for interrupts.

Note that the default treatment of an interrupt is not even required to be static. It is possible that an implementation or underlying operating system may have a mechanism that modifies the default treatment of an interrupt (or signal) while a program is running.

C.3(29) gives implementation advice as to examples of possible default treatments, but does not limit them:

> (1) The default treatment for an interrupt can be to keep the interrupt pending or to deliver it to an implementation-defined handler. Examples of actions that an implementation-defined handler is allowed to perform include aborting the partition, ignoring (i.e., discarding occurrences of) the interrupt, or queuing one or more occurrences of the interrupt for possible later delivery when a user-defined handler is attached to that interrupt.

Moreover, the intent of the standard is to allow user-specified handlers that are not parameterless protected procedures, as specified in C.3(26)

> Other forms of handlers are allowed to be supported, in which case, the rules of this subclause should be adhered to.

For example, it would be legal for an implementation to define a way to attach an ordinary procedure to an interrupt, or in the case of UNIX signals to attach ordinary procedures of several forms -- one with no parameter, one with a single, signal parameter, one with a signal parameter and a context parameter, etc.

This is emphasized in C.3.1(19):

> Notwithstanding what this subclause says elsewhere, the Attach_Handler and Interrupt_Handler pragmas are allowed to be used for other, implementation defined, forms of interrupt handlers.

and again in C.3.2(25):

> If implementation-defined forms of interrupt handler procedures are supported, such as protected procedures with parameters, then for each such form of a handler, a type analogous to Parameterless_Handler should be specified in a child package of Interrupts, with the same operations as in the predefined package Interrupts.

This means that if an implementation supports handlers or other interrupt treatments (whether default or user-specified) there will be situations in which the semantics of the operations defined in Ada.Interrupts for parameterless protected procedure handlers must be modified to take into account these other forms of treatments or handlers.

If Ada.Interrupts.Current_Handler or Ada.Interrupts.Exchange_Handler is called in a situation where the treatment in force for a given interrupt (whether default or user-specified) does not correspond to any parameterless protected procedure, the operation cannot return a value that designates a callable parameterless protected procedure.

Note that even if we were willing to require an implementation to create a "fake" Ada protected procedure, so as to be able to return a reference to a callable protected procedure, we have situations where the default treatment cannot be simulated by a protected procedure.

The intent of this section was that the value returned by Current_Handler or in Old_Handler may "represent" a handler or other treatment that is not a parameterless protected procedure, e.g., it might be an integer code for a default treatment or the address of some other kind of handler, unchecked-converted to the type

Parameterless_Handler. The notion was that such a value might not be usable for calling the handler directly from the application, but could meaningfully and safely be used to restore the old treatment or handler, by passing it back as New_Handler in a call to Exchange_Handler. This provides the capability for a user to safely install a handler, and then later restore the previous treatment, without needing to worry about whether the previous treatment is a parameterless protected procedure.

This intent is reflected in the first sentence of C.3.2(16), which says Current_Handler returns a value that "represents" the attached handler, rather than "designates". It is unfortunate that the word "designates" was inadvertently substituted in the second sentence, where the case of there being no user-defined handler is discussed.

Original intent aside, the question does raise a valid pragmatic issue. There are situations where it is desirable to allow the "cascading" of handlers, i.e., when a new handler is installed using Exchange_Handler, the new handler may use the previous handler (returned in the Old_Handler parameter) to call the previous hander -- for those situations where the previous handler was a parameterless protected procedure. However, to cascade handlers in this fashion, the application must know that there is a callable handler and the parameter profile of the handler.

It might be nice if there were a "portable" way for an application to determine whether the value returned in Old_Handler or by Current_Handler is one that can be dereferenced to call a parameterless protected procedure, as in the handler-cascading model. The standard itself does not explicitly specify how this can be determined, but it can be interpreted in a way that may suffice.

C.3.2(17) does require that the null access value can be used to specify the default treatment for an interrupt:

> The Attach_Handler procedure attaches the specified handler to the interrupt, overriding any existing treatment (including a user handler) in effect for that interrupt. If New_Handler is null, the default treatment is restored. ...

This binding interpretation extrapolates from the above to conclude that the value returned by Current_Handler, and the value returned in Old_Handler by Exchange_Handler, must be the null access value whenever the treatment that is in force for the given signal is the default treatment. It further extrapolates to conclude that the null access value must be returned whenever the treatment is not a user-installed handler. It follows that for implementations that support only parameterless protected procedures as handlers, these operations return only the null value and access values that can be used to call a parameterless protected procedure.

This provides a way to safely cascade user-installed handlers, provided the application uses only parameterless procedures as handlers.

This is a compromise. It does not provide the full capability that the original question hoped for; i.e., there is no way to "call" the default treatment. Likewise, if the implementation supports other forms of handlers, and the application uses them, there remains the possibility that the value returned by Current_Handler or in Old_Handler represents a handler that is not a parameterless protected procedure, and so it would be erroneous to call it without parameters. On the other hand, it would still be safe to use Exchange_Handler to restore such a handler.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0070**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **C.7.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0070 Abort_Task has a parameter of mode 'in'

Working Reference Number AI95-00101

Question

C.7.1(3) shows procedure Abort_Task taking a parameter of mode 'in out'. Is this correct? (No.)

Summary of Response

Task_Identification.Abort_Task takes a parameter of mode 'in':

```
procedure Abort_Task (T : in Task_Id);
```

Corrigendum Wording

In C.7.1(3) replace:

```
procedure Abort_Task   (T : in out Task_Id);
```

by:

```
procedure Abort_Task   (T : in Task_Id);
```

Discussion

Abort_Task does not modify its parameter, which is a Task_ID. Therefore, its parameter should be of mode 'in'.

Furthermore, if the parameter is of mode 'in out', then one cannot pass a function call or a constant. For example, Abort_Task(Current_Task) should be allowed. For another example, the following ought to be allowed:

```
type Task_ID_Array is array (Natural range <>) of Task_ID;
procedure Abort_Some_Tasks(Tasks: Task_ID_Array) is
begin
    for I in Tasks'range loop
        Abort_Task(Tasks(I));
    end loop;
end Abort_Some_Tasks;
...
Abort_Some_Tasks((This_Task_ID, That_Task_ID, The_Other_Task_ID));
```

Hence, this parameter should be of mode 'in'.

Note that Abort_Task is not analogous to Unchecked_Deallocation. After a call to an instance of Unchecked_Deallocation, the designated object ceases to exist, and any reference to it would be erroneous; therefore it makes sense for Unchecked_Deallocation to set the access object to null. However, after a call to Abort_Task, the task object continues to exist, and the task might even keep running for a while. Therefore, it does not make sense for Abort_Task to set its parameter to Null_Task_ID. Note that it is harmless to abort the same task twice -- either with an abort_statement, or with Abort_Task.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0071**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **C.7.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0071 Recursive use of task attributes

Working Reference Number AI95-00165

Question

C.7.2(16) says, in full, "The implementation shall perform each of the above operations for a given attribute of a given task atomically with respect to any other of the above operations for the same attribute of the same task."

Let us call an (attribute, task) pair a 'cell' for convenience. The atomicity requirement cannot be met if an operation on a cell recursively invokes an operation on the same cell: an operation cannot be atomic with respect to another operation embedded within itself.

What is the intent?

These operations can be recursive, because they perform finalization and assignment, which might invoke a user-defined Finalize or Adjust procedure, which might then recursively call the operation in question.

Here is an example of a recursive call to Set_Value:

```
with Ada.Finalization; use Ada.Finalization;
with Ada.Task_Attributes;
package An_Attr is
  type Attr is new Controlled with record
      N : Integer;
    end record;
  procedure Adjust(X : in out Attr);

  package Ops is new Ada.Task_Attributes(Attr,
        Initial_Value => (Controlled with 0));

end An_Attr;

with Ada.Text_IO; use Ada.Text_IO;
package body An_Attr is
  Depth : Natural := 0;

  procedure Adjust(X : in out Attr) is
  begin
    Put_Line((1..2*Depth => ' ') & "Adjust called");
    Depth := Depth + 1;
    if Depth <= 3 then
      Put_Line((1..2*Depth => ' ') & "calling Set_Value...");
      Ops.Set_Value((Controlled with Depth));
    end if;
  end Adjust;

end An_Attr;

with Ada.Finalization; use Ada.Finalization;
with An_Attr;
procedure A_Prog is
begin
  An_Attr.Ops.Set_Value((Controlled with 17)); -- One call
end A_Prog;
```

Finally, what happens if one of the operations of the package is concurrently executed with an access via an attribute handle? Is there an atomicity requirement on the latter as well?

Summary of Response

If the package Ada.Task_Attributes is instantiated with a controlled type and the controlled type has user-defined Adjust or Finalize operations that in turn access task attributes via instantiated interfaces of this

generic package, then a call of Set_Value of the instantiated package constitutes a bounded error. The call may perform as expected or it may result in a deadlock of the calling task and subsequently of the entire partition or of other tasks accessing task attributes.

Accesses via an Attribute_Handle (as obtained by calling the function Reference) are not subject to the atomicity requirement of C.7.2(16). Such accesses, if concurrent with each other or with the execution of any of the subprograms provided by the package, are erroneous.

Corrigendum Wording

Insert after C.7.2(13):

> For all the operations declared in this package, Tasking_Error is raised if the task identified by T is terminated. Program_Error is raised if the value of T is Null_Task_ID.

the new paragraph:

> *Bounded (Run-Time) Errors*
>
> If the package Ada.Task_Attributes is instantiated with a controlled type and the controlled type has user-defined Adjust or Finalize operations that in turn access task attributes by any of the above operations, then a call of Set_Value of the instantiated package constitutes a bounded error. The call may perform as expected or may result in forever blocking the calling task and subsequently some or all tasks of the partition.

Insert after C.7.2(15):

> If a value of Task_ID is passed as a parameter to any of the operations declared in this package and the corresponding task object no longer exists, the execution of the program is erroneous.

the new paragraph:

> Accesses to task attributes via a value of type Attribute_Handle are erroneous if executed concurrently with each other or with calls of any of the operations declared in package Task_Attributes.

Replace C.7.2(16):

> The implementation shall perform each of the above operations for a given attribute of a given task atomically with respect to any other of the above operations for the same attribute of the same task.

by:

> For a given attribute of a given task, the implementation shall perform the operations declared in this package atomically with respect to any of these operations of the same attribute of the same task. The granularity of any locking mechanism necessary to achieve such atomicity is implementation defined.

Response

A deadlock cannot be sensibly avoided when a recursive access via one of the interfaces of the package occurs to the same task attribute of the same task. Accesses via an Attribute_Handle are not subjected to the same atomicity and hence locking requirement. Therefore there are already dangerous situations and it seems inappropriate to impose a major performance penalty on some implementations in order to narrow only the already sufficiently rare deadlocking cases as much as possible.

A more liberal interpretation is recommended that allows implementations to choose the most appropriate lock granularity. A nested access to a task attribute from within a Finalize or Adjust procedure becomes a bounded error. Depending on the lock granularity, the initiating call of Set_Value will either deadlock or perform its nested accesses as expected. Concurrent accesses via Attribute_Handles are deemed erroneous.

The summary of this issue specifies these semantics in more detail.

Discussion

The atomicity of the operations will require some locking mechanism to prevent concurrent accesses to the same task attribute of a given task. C.7.2(16) seems to imply that an individual lock is provided for each

attribute of each task. However, obtaining a lock can be a rather expensive operation, particularly for implementations that utilize the locking primitives of an underlying operating system.

If the attribute does not involve controlled types with user-defined Adjust and Finalize routines, then a single run-time lock suffices to achieve the semantics of C.7.2(16).

If the attribute is of a controlled type or has components of a controlled type, then the implicitly invoked user-defined Adjust or Finalize routines are presently not forbidden to call on operations of this package for other task attributes or even, rather pathologically, for the same task attribute of the given task. Whatever locking strategy is applied, the latter will lead to a deadlock. The former can lead to a deadlock if the granularity of the lock is any larger than on single attributes of any task, e.g., a lock per task or a global run-time lock.

It seems unwise to require all implementations to provide a potentially expensive very fine-grained locking on attributes merely to guard against the fairly rare situation, in which

- a controlled type is chosen for a task attribute, and

- the Adjust or Finalize operation of the controlled type calls on operations of this package in turn to read or modify task attributes.

Also, the case of a recursive access to the same attribute of the same task will deadlock anyway.

The use of attribute handles is not protected by any atomicity requirement in the standard, so that their concurrent use must be deemed erroneous.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0072**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **D.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0072 Priority changes due to Set_Priority and Hold are not transitive

Working Reference Number AI95-00092

Question

D.1(21-22) say:

> During activation, a task being activated inherits the active priority of the its activator (see 9.2).
>
> During rendezvous, the task accepting the entry call inherits the active priority of the caller (see 9.5.3).

But this implies that if Set_Priority or Hold is called on a task, other tasks that are currently inheriting priority from the first task, would have to have their active priorities modified. Is this asynchronous priority inheritance the intent? (No.)

Summary of Response

If Set_Priority or Hold is called on a task, other tasks that are currently inheriting priority from the first task do not have their active priorities modified.

Corrigendum Wording

Replace D.1(21):

- During activation, a task being activated inherits the active priority of the its activator (see 9.2).

by:

- During activation, a task being activated inherits the active priority that its activator (see 9.2) had at the time the activation was initiated.

Replace D.1(22):

- During rendezvous, the task accepting the entry call inherits the active priority of the caller (see 9.5.3).

by:

- During rendezvous, the task accepting the entry call inherits the priority of the entry call (see 9.5.3 and D.4).

Discussion

D.1(21-22) seem to imply that asynchronous priority inheritance is required, meaning that when Set_Priority is called on a task, the active priority of other tasks must be modified. It was clearly not the intent to require asynchronous priority inheritance. Set_Priority is inherently asynchronous -- the task being affected may be doing anything when Set_Priority is called. However, we do not wish to require this asynchronous behavior to extend to *other* tasks -- asynchronous priority inheritance -- because it would be an implementation burden, and it is not clearly useful, given that priority inheritance is not uniformly transitive in all cases.

D.4(8-11) support the above as the "intent". In particular, D.4(11) says:

> When the base priority of a task is set (see D.5), if the task is blocked on an entry call, and the call is queued, the priority of the call is updated to the new active priority of the calling task. This causes the call to be removed from and then reinserted in the queue at the new active priority.

If Set_Priority on an entry caller were really intended to affect the active priority of a rendezvous-in-progress, then why would D.4(11) go to the trouble to say "and the call is queued"? This intent is also supported by the NOTE in D.11(17):

> If a task becomes held while waiting (as a caller) for a rendezvous to complete, the active priority of the accepting task is not affected.

There are two possible solutions:

> Alternative 1: Asynchronous priority inheritance does not happen. This is the interpretation given in the summary and wording above. In this alternative, if Set_Priority is applied to a task, then other tasks that are currently inheriting priority from the first task do not have their active priorities modified.
>
> Alternative 2: Asynchronous priority inheritance is not required, but an implementation may do it. In this alternative, when Set_Priority is applied to a task, it is implementation-defined whether or not other tasks that are currently inheriting priority from the first task have their active priorities modified.

Inheritance due to activation and rendezvous should be treated the same, and for rendezvous, it shouldn't make a difference whether the call is the trigger of an ATC or not. Both alternatives obey this principle.

The advantage of Alternative 1 is that it requires more uniformity across implementations. However, Alternative 2 seems to allow a fairly harmless implementation variation. Clearly, Alternative 2 is not harder to implement than Alternative 1. It is conceivable that Alternative 2 might be easier in some environments.

Note that either alternative allows an implementation to support asynchronous priority inheritance as a non-standard policy. Alternative 1 implies that if asynchronous priority inheritance is supported, the implementation must support two mechanisms, whereas Alternative 2 allows the asynchronous case to be the only one.

The implementation variation allowed by Alternative 2 is not entirely harmless. If a program is written assuming transitive priority inheritance, it could miss real-time deadlines when ported to an implementation that does not support transitive priority inheritance. When porting in the other direction, a working program could fail because of a violation of the ceiling rule in D.3(13):

> When a task calls a protected operation, a check is made that its active priority is not higher than the ceiling of the corresponding protected object; Program_Error is raised if this check fails.

In fact, Alternative 2 would allow an implementation to cause a "retroactive" violation of D.3(13). Presumably, the implementation would have to resolve this difficulty if it chose to implement asynchronous priority inheritance. Note that D.5(10) only talks about the task being directly affected, not other inheritors:

> Setting the task's base priority to the new value takes place as soon as is practical but not while the task is performing a protected action.

Presumably, the implementation would extend this deferral of priority changes to apply to the inheritors as well.

The discussion above refers to Set_Priority. The same arguments apply to Hold -- the whole point of defining Hold in terms of priorities was to avoid having to spell out all kinds of interactions between Hold and other tasking features.

Presumably, programs will not typically use both Hold and other features (like rendezvous) together. Therefore, the efficiency of Hold on a task in rendezvous doesn't matter. It's just important that there be little or no distributed overhead (in either direction).

Note that the issue does not arise for protected entry calls (the case in D.1(23)), because ceiling priorities can never change.

Given the advantages of implementation uniformity, Alternative 1 is chosen. Implementations that wish to support asynchronous priority inheritance must do so via a non-standard policy.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0073**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **D.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0073 Pragma Locking_Policy cannot be in a program unit

Working Reference Number AI95-00091

Question

D.3(6) says:

> If no Locking_Policy pragma *appears in* any of the program units comprising a partition, the locking policy for that partition, as well as the effect of specifying either a Priority or Interrupt_Priority pragma for a protected object, are implementation defined. [Emphasis added.]

But Locking_Policy is a configuration pragma, and configuration pragmas do not "appear in" program units. What is the intent?

Summary of Response

If no Locking_Policy pragma applies to any of the program units comprising a partition, the locking policy for that partition, as well as the effect of specifying either a Priority or Interrupt_Priority pragma for a protected object, are implementation defined.

Corrigendum Wording

In D.3(6) replace:

> If no Locking_Policy pragma appears in any of the program units comprising a partition, the locking policy for that partition, as well as the effect of specifying either a Priority or Interrupt_Priority pragma for a protected object, are implementation defined.

by:

> If no Locking_Policy pragma applies to any of the program units comprising a partition, the locking policy for that partition, as well as the effect of specifying either a Priority or Interrupt_Priority pragma for a protected object, are implementation defined.

Discussion

The intent is as stated under "wording".

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0074**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **D.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0074 Number of queuing policies defined

Working Reference Number AI95-00068

Question

D.4(1) says "It also defines one such policy." But the language defines two such policies. What is meant here?

Summary of Response

D.4 defines *two* language-defined policies.

Corrigendum Wording

Replace D.4(1):

> This clause specifies a mechanism for a user to choose an entry *queuing policy*. It also defines one such policy. Other policies are implementation defined.

by:

> This clause specifies a mechanism for a user to choose an entry *queuing policy*. It also defines two such policies. Other policies are implementation defined.

Response

The wording for D.4(1) is hereby corrected.

Discussion

This is just an editing error. Obviously, the language defines two policies (FIFO_Queuing and Priority_Queuing).

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0075**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **D.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0075 Priority changes in abortable part

Working Reference Number AI95-00205

Question

If Set_Priority is called in the abortable part, is the priority of the triggering entry call updated? (No.) D.4(10) does not apply (because the base priority of the task is set), so it appears that the update must occur (because either D.4(10) or D.4(11) must apply). But there is a validation test which requires otherwise.

Summary of Response

For the priority queuing policy, if the base priority is changed in an abortable part while a triggering entry call is queued, the priority of the entry call is not affected. (That is, the rule of D.4(10) applies.)

Corrigendum Wording

Replace D.4(10):

- After a call is first queued, changes to the active priority of a task do not affect the priority of the call, unless the base priority of the task is set.

by:

- After a call is first queued, changes to the active priority of a task do not affect the priority of the call, unless the base priority of the task is set while the task is blocked on an entry call.

Discussion

The language designers did not want D.4(11) to apply in this case. This is supported by the use of the phrase "if the task is blocked on an entry call" in D.4(11) -- the task making a triggering entry call is not blocked. It is further supported by the lengthy discussion in the AARM on this very topic (AARM D.4(11.a-11.f)).

However, the language designers failed to note that the wording of D.4(10) prevents it from applying in this case as well. The intent was that D.4(10) would apply to all entry calls unless D.4(11) applied.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0076**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **D.7**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0076 Pragma Restrictions(Max_Tasks, Max_Asynchronous_Select_Nesting)

Working Reference Number AI95-00067

Question

The Real-Time Systems annex says in D.7(15) (of the Max_Tasks and Max_Asynchronous_Select_Nesting restrictions):

> If the following restrictions are violated, the behavior is implementation defined. If an implementation chooses to detect such a violation, Storage_Error should be raised.

The Safety and Security annex says in H.4(2):

> The following restrictions, the same as in D.7, apply in this Annex: No_Task_Hierarchy, No_Abort_Statement, No_Implicit_Heap_Allocation, Max_Task_Entries is 0, Max_Asynchronous_Select_Nesting is 0, and Max_Tasks is 0. The last three restrictions are checked prior to program execution.

Suppose an implementation complies with both annexes. Is the following example legal? (No.)

```
pragma Restrictions (Max_Tasks => 0);
procedure main is
begin
   if false then
      declare
         task x is -- Legal?  (No.)
         ...
   end if;
end main;
```

What does it mean that the restriction is "checked prior to program execution"?

Summary of Response

For a pragma Restrictions(Max_Tasks => 0), task creation is illegal, for both the Real-Time Systems annex and the Safety and Security annex. Similarly, for a pragma Restrictions(Max_Asynchronous_Select_Nesting => 0), asynchronous_selects are illegal, for both of these annexes.

Corrigendum Wording

Delete D.7(15):

> If the following restrictions are violated, the behavior is implementation defined. If an implementation chooses to detect such a violation, Storage_Error should be raised.

Replace D.7(17):

> Max_Storage_At_Blocking
>
> Specifies the maximum portion (in storage elements) of a task's Storage_Size that can be retained by a blocked task.

by:

> Max_Storage_At_Blocking
>
> Specifies the maximum portion (in storage elements) of a task's Storage_Size that can be retained by a blocked task. If an implementation chooses to detect a violation of this restriction, Storage_Error should be raised; otherwise, the behavior is implementation defined.

Replace D.7(18):

Max_Asynchronous_Select_Nesting
Specifies the maximum dynamic nesting level of asynchronous_selects. A value of zero prevents the use of any asynchronous_select.

by:

Max_Asynchronous_Select_Nesting
Specifies the maximum dynamic nesting of asynchronous_selects. A value of zero prevents the use of any asynchronous_select and, if a program contains an asynchronous_select, it is illegal. If an implementation chooses to detect a violation of this restriction for values other than zero, Storage_Error should be raised; otherwise, the behavior is implementation defined.

Replace D.7(19):

Max_Tasks
Specifies the maximum number of task creations that may be executed over the lifetime of a partition, not counting the creation of the environment task.

by:

Max_Tasks
Specifies the maximum number of task creations that may be executed over the lifetime of a partition, not counting the creation of the environment task. A value of zero prevents any task creation and, if a program contains a task creation, it is illegal. If an implementation chooses to detect a violation of this restriction for values other than zero, Storage_Error should be raised; otherwise, the behavior is implementation defined.

Response

An implementation conforming to the Safety and Security annex must support a pragma Restrictions(Max_Tasks => E), where E is a static expression whose value is zero. If such a pragma applies to a given compilation unit, then for an implementation conforming to the Real-Time Systems or Safety and Security annex (or both), the compilation unit is illegal if it contains an object_declaration or allocator, where the type of the created object is a task type, or is a composite type with some subcomponent type that is a task type.

An implementation conforming to the Safety and Security annex must support a pragma Restrictions(Max_Asynchronous_Select_Nesting => E), where E is a static expression whose value is zero. If such a pragma applies to a given compilation unit, then for an implementation conforming to the Real-Time Systems or Safety and Security annex (or both), the compilation unit is illegal if it contains an asynchronous_select.

Discussion

The intent is that it should be possible for a single implementation to comply with all of the Specialized Needs Annexes. Therefore, there cannot be contradictory requirements in two different Specialized Needs Annexes.

"Max_Tasks is 0" should be interpreted to mean that a static expression is given, and its value is zero. Clearly, we cannot require compile-time detection unless the expression is static. But we don't want to interpret "is 0" to mean that it must be a literal with value zero, or a literal containing exactly the character "0" -- such a restriction would be inconsistent with other rules that need compile-time-known values. Thus, an expression like "00" or "1 - 1" should be allowed.

What does it mean that the restriction is "checked prior to program execution"? This could be interpreted to mean a legality check, or could be interpreted to mean that a diagnostic message, such as a warning message, must be given at compile time, but the program is still legal, and may be executed.

What exactly is being checked at compile time? The only reasonable interpretation would seem to be to check for the existence of an object_declaration or allocator, where the type of the created object is a task

type, or is a composite type with some subcomponent type that is a task type. Thus, for example, a null array of tasks would fail this check.

The following possible interpretations exist:

> 1. If the implementation supports the Real-Time annex, the example program is legal. If the implementation supports the Safety and Security annex, the example is illegal. We reject this interpretation, because it constitutes a contradiction between the two annexes.
>
> 2. If the expression is statically zero, then the example is illegal, if either annex is supported. This is the interpretation chosen.
>
> 3. If the expression is statically zero, then the example is legal. However, if the implementation conforms to the Safety and Security annex, then it must issue a warning message. There is some precedent for requiring warning messages -- see 2.8(13). This seems like a reasonable interpretation. However, it seems better to simply declare the program illegal -- a warning message doesn't have any particular value here.
>
> 4. Delete the requirement to check the restriction before run time. We reject this, because it does not satisfy the needs of the Safety and Security annex -- namely, to know at compile time whether the program is wrong.

Similar arguments apply to Max_Asynchronous_Select_Nesting.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0077**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **D.11; J.7.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0077 Accept body not defined

Working Reference Number AI95-00111

Question

The term "accept body" is used in D.11(18), J.7.1(16), and J.7.1(20). It is not defined anywhere.

Summary of Response

The phrase "accept body" should be replaced by other wording.

Corrigendum Wording

Replace D.11(18):

- If a task becomes held while waiting in a selective_accept, and a entry call is issued to one of the open entries, the corresponding accept body executes. When the rendezvous completes, the active priority of the accepting task is lowered to the held priority (unless it is still inheriting from other sources), and the task does not execute until another Continue.

by:

- If a task becomes held while waiting in a selective_accept, and an entry call is issued to one of the open entries, the corresponding accept_alternative executes. When the rendezvous completes, the active priority of the accepting task is lowered to the held priority (unless it is still inheriting from other sources), and the task does not execute until another Continue.

Replace J.7.1(16):

Interrupt entry calls may be implemented by having the hardware execute directly the appropriate accept body. Alternatively, the implementation is allowed to provide an internal interrupt handler to simulate the effect of a normal task calling the entry.

by:

Interrupt entry calls may be implemented by having the hardware directly execute the appropriate accept_statement. Alternatively, the implementation is allowed to provide an internal interrupt handler to simulate the effect of a normal task calling the entry.

Replace J.7.1(20):

NOTES

1 Queued interrupts correspond to ordinary entry calls. Interrupts that are lost if not immediately processed correspond to conditional entry calls. It is a consequence of the priority rules that an accept body executed in response to an interrupt can be executed with the active priority at which the hardware generates the interrupt, taking precedence over lower priority tasks, without a scheduling action.

by:

NOTES

1 Queued interrupts correspond to ordinary entry calls. Interrupts that are lost if not immediately processed correspond to conditional entry calls. It is a consequence of the priority rules that an accept_statement executed in response to an interrupt can be executed with the active priority at which the hardware generates the interrupt, taking precedence over lower priority tasks, without a scheduling action.

Discussion

This was an editing error; "accept body" was defined in a draft of the standard.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0078**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **E.2; E.2.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0078 An RCI unit can be a library subprogram

Working Reference Number AI95-00048

Question

The rules for pragma Shared_Passive (E.2.1(3)), pragma Remote_Types (E.2.2(3)), and pragma Remote_Call_Interface (E.2.3(3)) seem to allow them to apply to any library unit. However, the definitions in E.2(4) seem to imply that only packages and generic packages are allowed:

> A library package or generic library package is called a shared passive library unit if a Shared_Passive pragma applies to it. A library package or generic library package is called a remote types library unit if a Remote_Types pragma applies to it. A library package or generic library package is called a remote call interface if a Remote_Call_Interface pragma applies to it. A normal library unit is one to which no categorization pragma applies.

What is the intent? Can a subprogram or generic subprogram be a shared passive or remote types library unit? (No.) Can a subprogram or generic subprogram be a remote call interface library unit? (Yes.)

Summary of Response

A shared passive or remote types library unit must be a package or generic package, not a subprogram or generic subprogram. However, a remote call interface library unit may be a package, generic package, subprogram, or generic subprogram.

Corrigendum Wording

Replace E.2(4):

> A library package or generic library package is called a *shared passive* library unit if a Shared_Passive pragma applies to it. A library package or generic library package is called a *remote types* library unit if a Remote_Types pragma applies to it. A library package or generic library package is called a *remote call interface* if a Remote_Call_Interface pragma applies to it. A *normal library unit* is one to which no categorization pragma applies.

by:

> A library package or generic library package is called a *shared passive* library unit if a Shared_Passive pragma applies to it. A library package or generic library package is called a *remote types* library unit if a Remote_Types pragma applies to it. A library unit is called a *remote call interface* if a Remote_Call_Interface pragma applies to it. A *normal library unit* is one to which no categorization pragma applies.

Replace E.2.3(7):

> A *remote call interface (RCI)* is a library unit to which the pragma Remote_Call_Interface applies. A subprogram declared in the visible part of such a library unit is called a *remote subprogram*.

by:

> A *remote call interface (RCI)* is a library unit to which the pragma Remote_Call_Interface applies. A subprogram declared in the visible part of such a library unit, or declared by such a library unit, is called a *remote subprogram*.

Replace E.2.3(9):

> In addition, the following restrictions apply to the visible part of an RCI library unit:

by:

> In addition, the following restrictions apply to an RCI library unit:

Replace E.2.3(10):

- it shall not contain the declaration of a variable;

by:

- its visible part shall not contain the declaration of a variable;

Replace E.2.3(11):

- it shall not contain the declaration of a limited type;

by:

- its visible part shall not contain the declaration of a limited type;

Replace E.2.3(12):

- it shall not contain a nested generic_declaration;

by:

- its visible part shall not contain a nested generic_declaration;

Replace E.2.3(13):

- it shall not contain the declaration of a subprogram to which a pragma Inline applies;

by:

- it shall not be, nor shall its visible part contain, the declaration of a subprogram to which a pragma Inline applies;

Replace E.2.3(14):

- it shall not contain a subprogram (or access-to-subprogram) declaration whose profile has an access parameter, or a formal parameter of a limited type unless that limited type has user-specified Read and Write attributes;

by:

- it shall not be, nor shall its visible part contain, a subprogram (or access-to-subprogram) declaration whose profile has an access parameter, or a formal parameter of a limited type unless that limited type has user-specified Read and Write attributes;

Replace E.2.3(19):

If a pragma All_Calls_Remote applies to a given RCI library package, then the implementation shall route any call to a subprogram of the RCI package from outside the declarative region of the package through the Partition Communication Subsystem (PCS); see E.5. Calls to such subprograms from within the declarative region of the package are defined to be local and shall not go through the PCS.

by:

If a pragma All_Calls_Remote applies to a given RCI library unit, then the implementation shall route any call to a subprogram of the RCI unit from outside the declarative region of the unit through the Partition Communication Subsystem (PCS); see E.5. Calls to such subprograms from within the declarative region of the unit are defined to be local and shall not go through the PCS.

Discussion

The wording is ambiguous, and the intent is unclear. Clearly, shared passive subprograms and remote types subprograms make no sense. However, RCI subprograms make sense. We have two choices:

Option 1: A shared passive, remote types, or remote call interface library unit must be a package or generic package, not a subprogram or generic subprogram.

Option 2 (as in the summary above): A shared passive or remote types library unit must be a package or generic package, not a subprogram or generic subprogram. However, a remote call interface library unit may be a package, generic package, subprogram, or generic subprogram. A main subprogram may be an RCI unit.

The argument for Option 1 is that RCI subprograms are not particularly important, the original designers apparently intended to allow only packages, and the wording changes are easier if this choice is chosen.

The argument for Option 2 is that RCI subprograms make sense (given that library subprograms are allowed in the first place), and it would seem like an ugly and arbitrary restriction to disallow them.

We choose Option 2.

Note that, given the above wording changes, 10.2(29) without changes implies that main subprograms that are RCI subprograms must be supported. We see no implementation difficulty in that.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0079**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **E.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0079 What is the meaning of "same representation" in all partitions?

Working Reference Number AI95-00208

Question

The implementation requirement E.2(13) says:

> For a given library-level type declared in a preelaborated library unit or in the declaration of a remote types or remote call interface library unit, the implementation shall choose the same representation for the type upon each elaboration of the type's declaration for different partitions of the same program.

This seems overly restrictive. It means that the standard supports heterogeneous distributed systems only if the implementation manages to use the same representation for a type on all nodes. Is this intended? (No.)

Summary of Response

E.2(13) requires the "same representation" in all partitions. This requirement prevents heterogeneous distributed systems, and is not needed, so it is deleted.

Corrigendum Wording

Delete E.2(13):

> For a given library-level type declared in a preelaborated library unit or in the declaration of a remote types or remote call interface library unit, the implementation shall choose the same representation for the type upon each elaboration of the type's declaration for different partitions of the same program.

Discussion

The requirement for the same representation for types in shareable library packages is over-specification. What is required is that the types have similar semantics and have consistent values when remotely accessed. While requiring the same representation insures that the requirement is met, it prevents many possible heterogeneous systems. For instance, the byte order of integer types may be different on different processors, but this has no effect on the semantics of the types.

If an implementation uses a representation independent format for its communication between partitions, heterogeneous distributed systems on processors with differing numeric formats can be supported. (Note that the storage element stream for a scalar type is implementation-defined.) Such an implementation could correctly support all of the semantics of Annex E.

Such heterogeneous systems have been used in avionics to insure that answers calculated are independent of implementation and processor errors. This is done by calculating answers on several different processors (each with its own executable code), and comparing them. Ada should support the construction of such systems with each processor running one or more active partitions.

Furthermore, there seems to be no reason to require the rule even for homogeneous systems. The rule essentially says that the easiest implementation is required. If an Ada implementation has a reason to go through the extra work to support multiple representations, there seems to be no reason for the standard to get in its way.

If an implementation does support different representations in different active partitions, we believe it is best to leave the exact semantics to the implementation. We expect that implementations will do what is necessary to insure that the semantics of types with different representations is similar and meaningful in each partition.

Therefore, the rule requiring the same representation in each partition is not needed, and is deleted.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0080**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **E.2.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0080 Access types declared in shared passive generic packages

Working Reference Number AI95-00003

Question

E.2.1(7) states that a shared passive library unit "shall not contain a library-level declaration of an access type that designates a class-wide type, task type, or protected type with entry_declarations; if the shared passive library unit is generic, it shall not contain a declaration for such an access type unless the declaration is nested within a body other than a package_body."

This allows such an access type in a block_statement in the sequence_of_statements of the body of a package, but not of a generic package (since a block_statement is not a "body other than a package_body"). Is this intended? (No.)

Summary of Response

A declaration of an access type that designates a class-wide type, task type, or protected type with entry_declarations is allowed within a block_statement in the sequence_of_statements of a generic shared passive package.

Corrigendum Wording

Replace E.2.1(7):

- it shall not contain a library-level declaration of an access type that designates a class-wide type, task type, or protected type with entry_declarations; if the shared passive library unit is generic, it shall not contain a declaration for such an access type unless the declaration is nested within a body other than a package_body.

by:

- it shall not contain a library-level declaration of an access type that designates a class-wide type, task type, or protected type with entry_declarations.

Discussion

The intent of the part of E.2.1(7) after the semicolon is to forbid certain access types in a generic shared passive package, unless the access type is declared within a master. 3.10.2(20) says:

> For determining whether one level is statically deeper than another when within a generic package body, the generic package is presumed to be instantiated at the same level as where it was declared; run-time checks are needed in the case of more deeply nested instantiations.

This implies that the part of E.2.1(7) after the semicolon is redundant, except in the case of block_statements. The block_statement case was not intended to be forbidden (and is not forbidden in the non-generic case). Thus, these words should be deleted.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0081**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0081 Conversions to types derived from remote access types

Working Reference Number AI95-00004

Question

NOTE 3.4(31) says, "If the parent type is an access type, then the parent and the derived type share the same storage pool...". E.2.2(17) says:

> The Storage_Pool and Storage_Size attributes are not defined for remote access-to-class-wide types; the expected type for an allocator shall not be a remote access-to-class-wide type; a remote access-to-class-wide type shall not be an actual parameter for a generic formal access type;

This seems to imply that a remote access type has no storage pool, which is confirmed by AARM E.2.2(17.a):

> Reason: All three of these restrictions are because there is no storage pool associated with a remote access-to-class-wide type.

However, E.2.2(17) allows allocators for types derived from remote access types. How can an allocator work for a type that has no storage pool?

Summary of Response

Notwithstanding the rule in 3.10(7), a remote access type (unlike other access types) has no associated storage pool. If a type is derived from a remote access type, then the derived type is also a remote access type.

Corrigendum Wording

Replace E.2.2(9):

> An access type declared in the visible part of a remote types or remote call interface library unit is called a *remote access type*. Such a type shall be either an access-to-subprogram type or a general access type that designates a class-wide limited private type.

by:

> An access type declared in the visible part of a remote types or remote call interface library unit is called a *remote access type*. Such a type shall be either an access-to-subprogram type or a general access type that designates a class-wide limited private type. A type that is derived from a remote access type is also a remote access type.

Response

Notwithstanding the rule in 3.10(7), a remote access type (unlike other access types) has no associated storage pool. If a type is derived from a remote access type, then the derived type is also a remote access type, and hence also has no associated storage pool. Thus, type conversions are allowed between such types (see E.2.2(11,15)). The restrictions in E.2.2(10-17) apply to types derived from remote access types.

Discussion

Normally, a derived access type has the same storage pool as its parent. See 8652/0012 (AI-00062), which confirms NOTE 3.4(31). However, the intent of E.2.2(17) is that a remote access type has no storage pool. Therefore, a type derived from a remote access type cannot have a storage pool, either. Querying 'Storage_Pool and 'Storage_Size should be illegal by E.2.2(17). Similarly, allocators should be illegal.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0082**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0082 Definition of remote access type

Working Reference Number AI95-00164

Question

An interpretation of E.2.2(9) would deny object-oriented programming methodology to distributed Ada programmers by not permitting a remote-access-to-class-wide (RACW) type to designate a class-wide private extension of limited private type. If this interpretation holds then the following example is illegal:

```
package RT is
   pragma Remote_Types;

   type Root_Object is abstract tagged limited private;

   type New_Object is new Root_Object with private;

   -- Primitive dispatched procedures of New_Object.
   procedure Operation_1 (Obj : access New_Object; Z : Integer);

private
   type Root_Object is abstract tagged limited null record;
   type New_Object is new Root_Object with null record;
end RT;

with RT;
package RCI is
    pragma Remote_Call_Interface;

    type New_Access is access all RT.New_Object'Class;
    --   ^^^^^^^^^^
    --     illegal

    procedure Register (New_Obj : New_Access);
end;
```

Also, while limitedness of the target type is clearly needed, there seems to be no reason why the target type needs to be private.

Summary of Response

E.2.2(9) is interpreted to permit a remote access-to-class-wide type to designate a class-wide private extension of a limited private type.

Corrigendum Wording

Replace E.2.2(9):

> An access type declared in the visible part of a remote types or remote call interface library unit is called a *remote access type*. Such a type shall be either an access-to-subprogram type or a general access type that designates a class-wide limited private type.

by:

> An access type declared in the visible part of a remote types or remote call interface library unit is called a *remote access type*. Such a type shall be:
>
> - an access-to-subprogram type, or
> - a general access type that designates a class-wide limited private type or a class-wide private type extension all of whose ancestors are either private type extensions or limited private types.

Discussion

There were two issues raised:

application of RACW to private extension of limited private type;

removal of restriction on RACW to private types.

Regarding the first issue, the intent of E.2.2(9) is not to exclude private extensions. The conclusion that E.2.2(9) denies distributed object programming seems unwarranted given that typically the designated type is most naturally extended in the body of a package where the distributed/remote object is declared.

Regarding the second issue, if E.2.2(9) is relaxed to allow the type to be completed in the visible part of the package this would provide additional capability only to those objects that are to be accessed locally. Thus, there is no significant gain in a distributed application. The requirement that the designated type of the remote access-to-class-wide type be limited private is consistent with that placed upon a file type since in each case they both provide a handle to some external object.

Retaining the restriction that this paragraph apply to only private types (and their extensions) ensures the least surprise to developers when non-distributed software modules are subsequently inserted into a distributed environment.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0083**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **E.2.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0083 User-defined Read and Write attributes

Working Reference Number AI95-00047

Question

E.2.2(14) says, "... the types of all the noncontrolling formal parameters shall have Read and Write attributes."

By 13.13.2(2), this is vacuously true. Do you mean *user-specified* Read and Write attributes, as suggested by the note in E.2.2(18)? (That would be a strange requirement for, say, a parameter of type Integer, but the obvious alternative interpretation also seems strange.)

Summary of Response

Consider a remote access-to-classwide type, whose corresponding specific type is T, and a primitive subprogram P of T. For each non-controlling parameter of P, if its type is limited, it must have user-defined 'Read and 'Write operations.

Corrigendum Wording

Replace E.2.2(14):

- The primitive subprograms of the corresponding specific limited private type shall only have access parameters if they are controlling formal parameters; the types of all the non-controlling formal parameters shall have Read and Write attributes.

by:

- The primitive subprograms of the corresponding specific limited private type shall only have access parameters if they are controlling formal parameters; each non-controlling formal parameter shall have either a nonlimited type or a type with Read and Write attributes specified via an attribute_definition_clause;

Discussion

The intent of E.2.2(14) is to require that every non-controlling parameter have 'Read and 'Write operations that can be called. Although 'Read and 'Write always *exist* by 13.13.2(2), it is illegal to call them for limited types, unless they are user specified, by 13.13.2(36). (The reason for this circumlocution was to avoid a generic contract model problem.)

To see the reason for the rule, consider the following example:

```
package Pure_Pkg is
   type Lim is limited
      record
         ...
      end record;
   for Lim'Read use ...;
   for Lim'Write use ...;

   type T is abstract tagged limited private;
   procedure P(Controlling_Param: access T;
               Noncontrolling_Param: Lim) is abstract;
private
   ...
end Pure_Pkg;

with Pure_Pkg; use Pure_Pkg;
package RCI is
   pragma Remote_Call_Interface;

   type Remote_Access_To_Classwide is access all T'Class;
```

```
end RCI;
```

Now we declare an object of the remote access-to-classwide type:

```
X: Remote_Access_To_Classwide := ...;
```

X might point to an object in some other partition.

Now we write a dispatching call:

```
L: Lim;
...
P(X, L);
```

This will do a remote call to whatever partition contains the object designated by X. We need to transfer the value of L to that partition, which would be impossible if Lim did not have user-defined Read and Write attributes.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0084**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **E.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0084 Version and Body_Version attributes

Working Reference Number AI95-00104

Question

Two questions:

E.3(4) says:

> P'Body_Version
>
> > Yields a value of the predefined type String that identifies the version of the compilation unit that contains the body (but not any subunits) of the program unit.

What if the program unit has no body?

E.3(5) says:

> The version of a compilation unit changes whenever the version changes for any compilation unit on which it depends semantically. The version also changes whenever the compilation unit itself changes in a semantically significant way. It is implementation defined whether there are other events (such as recompilation) that result in the version of a compilation unit changing.

First of all, it is not clear what "semantically significant" means. Second of all, the "implementation defined" part seems to leave a huge loophole; an implementation could change the version on every clock tick (at run time), which would mean the Version and Body_Version attributes would return a different value every time, which would make them useless. What is the intent?

Summary of Response

If P is not a library unit, and P has no completion, then P'Body_Version returns the Body_Version of the innermost program unit enclosing the declaration of P. If P is a library unit, and P has no completion (which can be detected at compile time), then P'Body_Version returns a value that is different from Body_Version of any version of P that has a completion.

E.3(5) is replaced with:

The version of a compilation unit changes whenever the compilation unit changes in a semantically significant way. This International Standard does not define the exact meaning of "semantically significant". It is also unspecified whether there are other events (such as recompilation) that result in the version of a compilation unit changing.

Corrigendum Wording

Replace E.3(5):

> The *version* of a compilation unit changes whenever the version changes for any compilation unit on which it depends semantically. The version also changes whenever the compilation unit itself changes in a semantically significant way. It is implementation defined whether there are other events (such as recompilation) that result in the version of a compilation unit changing.

by:

> The *version* of a compilation unit changes whenever the compilation unit changes in a semantically significant way. This International Standard does not define the exact meaning of "semantically significant". It is also unspecified whether there are other events (such as recompilation) that result in the version of a compilation unit changing.

> If P is not a library unit, and P has no completion, then P'Body_Version returns the Body_Version of the innermost program unit enclosing the declaration of P. If P is a library unit, and P has no completion, then P'Body_Version returns a value that is different from Body_Version of any version of P that has a completion.

Discussion

It should not be an error to query P'Body_Version when P has no body, because:

> 1. The purpose of P'Body_Version is to distinguish different implementations of P. If P has a body, that is a different implementation of P than if P does not have a body. P'Body_Version should return a different value in those two cases, not give an error. The client cares whether the implementation of P has changed; the client should not have to know whether or not that implementation involves a body.
>
> 2. The "error" is not detectable at compile time, in general. In particular, if P is not a library unit, one cannot tell from the declaration of P whether or not it has a body.

The summary uses the term "completion" to account for the fact that there might be a pragma Import instead of a body.

As to the second question, we choose to leave "semantically significant" vague, and trust implementations to do something sensible. The version should not change at the drop of a hat; in any given implementation, there should at least be some way of ensuring that execution of identical source code produces an identical version at run time. Some sensible implementations are:

> - The "traditional" (Ada 83) program library model: It would make sense for the version to be a time stamp representing the time of compilation. If the programmer recompiles a compilation unit, it will get a new version. The mechanism for ensuring identical versions is, "Don't recompile it".
>
> - A "source-based" model: The version is a combination of the time stamp of the source of the compilation unit itself, plus all compilation units upon which it depends semantically. If the compiler can guarantee that the same source always produces the same object code, which is usual, then the version could change if and only if the user edits the source files (whether or not any changes were actually made). The mechanism for ensuring identical versions is, "Don't edit the source files".
>
> - An optimization of the source-based model: The version is a "hash value" calculated from the source code of the compilation unit itself, plus all compilation units upon which it depends semantically. Comments are deleted before calculating the hash value. The mechanism for ensuring identical versions is, "Don't edit the source files, except to modify comments."
>
> - A different optimization: The last 20 versions of every file are remembered by the implementation. If the current version is identical to one of the remembered ones, then it gets the same version. Otherwise, it gets a new version.

We don't want to require that identical source code always produces identical versions at run time. However, an implementation should provide *some* way of producing identical versions at run time when the source code hasn't changed. All of the above-mentioned possible implementations have this property. One can imagine much more sophisticated mechanisms, and we don't want to forbid them.

In any case, it seems reasonable that if the object code changes, the version should change. We state this "only" as advice, because the International Standard has no formal concept of object code. In particular, there is no standard way of knowing which pieces of object code belong to which compilation units.

The intent is that if the user does something semantically neutral, like adding a comment, then an implementation should be *allowed* to keep the version the same. In order to facilitate such "smart recompilation" strategies, we remove the phrase "implementation defined" from E.3(5), so that implementations need not document the exact cases when the version changes.

Note that we remove the wording, "The version of a compilation unit changes whenever the version changes for any compilation unit on which it depends semantically." from E.3(5), because a compiler might be able to prove that whatever change was made to the compilation unit on which it depends semantically is irrelevant.

In summary, in our view, the version of a compilation unit should change when its generated code changes. The version of a compilation unit should change when the version changes for a compilation unit upon which it depends semantically, if the change has a semantically significant effect on the first compilation unit. There may be other situations that also cause the version to change, but the implementation should provide a way to ensure that the version does not change if the compilation unit and the compilation units upon which it depends semantically do not change.

Note that if X is a renaming declaration (not a renaming-as-body), then X'Version and X'Body_Version refer to the versions of the renamed entities.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0085**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Omission**
REFERENCES IN DOCUMENT: **E.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0085 Returning remote class-wide values

Working Reference Number AI95-00215

Question

The rule of E.4(18) requires a check for the actual parameter of a remote subprogram call with a formal parameter of a class-wide type. This check is to prevent the passage of objects whose type is not a "communicable" type. However, no check is required for a function returning a class-wide object. Therefore, a function can return an object that is not of a "communicable" type. Was this intended? (No.)

Summary of Response

A check is made on the result of a remote function call that returns a class-wide type that it does not violate the conditions described in section E.4(18).

Corrigendum Wording

Replace E.4(18):

> In a remote subprogram call with a formal parameter of a class-wide type, a check is made that the tag of the actual parameter identifies a tagged type declared in a declared-pure or shared passive library unit, or in the visible part of a remote types or remote call interface library unit. Program_Error is raised if this check fails.

by:

> In a remote subprogram call with a formal parameter of a class-wide type, a check is made that the tag of the actual parameter identifies a tagged type declared in a declared-pure or shared passive library unit, or in the visible part of a remote types or remote call interface library unit. Program_Error is raised if this check fails. In a remote function call which returns a class-wide type, the same check is made on the function result.

Discussion

The purpose of the rule E.4(18) is to prevent the passage of objects which have types that are not "communicable" types. (That is, types which are not known to the other partitions.) Normally this is enforced at compile-time, but with class-wide types, the check needs to be a run-time check.

The rule would not be helpful if there was a way to pass objects of types that are not "communicable". However, exactly that can be done by returning such an object from a function. Clearly, a check needs to be made on such results as well.

Note that the return statement in the function itself cannot know whether or not it was called remotely. Therefore, the check must be made by the remote function call return code when the object is marshalled to be returned to the caller.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0086**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **E.4**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0086 Shared variables in Shared_Passive?

Working Reference Number AI95-00159

Question

There is no task rendezvous between two partitions, and protected entries are disallowed in Shared_Passive packages, so how can two actions of reading/updating variables declared in a Shared_Passive package performed on two different partitions be sequential as defined by 9.10(11)?

Summary of Response

For the purposes of the shared variables rules in 9.10, with respect to shared variables in shared passive partitions, a synchronous remote procedure call is considered to be part of the execution of the calling task.

For an asynchronous RPC, the call signals the start of the remote body, but the body then proceeds in parallel, and thus does not signal the next action of the calling task.

Corrigendum Wording

Insert after E.4(20):

> The implementation of remote subprogram calls shall conform to the PCS interface as defined by the specification of the language-defined package System.RPC (see E.5). The calling stub shall use the Do_RPC procedure unless the remote procedure call is asynchronous in which case Do_APC shall be used. On the receiving side, the corresponding receiving stub shall be invoked by the RPC-receiver.

the new paragraph:

> With respect to shared variables in shared passive library units, the execution of the corresponding subprogram body of a synchronous remote procedure call is considered to be part of the execution of the calling task. The execution of the corresponding subprogram body of an asynchronous remote procedure call proceeds in parallel with the calling task and does not signal the next action of the calling task (see 9.10).

Discussion

9.10 says:

> (2) Separate tasks normally proceed independently and concurrently with one another. However, task interactions can be used to synchronize the actions of two or more tasks to allow, for example, meaningful communication by the direct updating and reading of variables shared between the tasks. The actions of two different tasks are synchronized in this sense when an action of one task signals an action of the other task; an action A1 is defined to signal an action A2 under the following circumstances:
>
> (3) If A1 and A2 are part of the execution of the same task, and the language rules require A1 to be performed before A2;
>
> ...
>
> (7) If A1 is the action of issuing an entry call, and A2 is part of the corresponding execution of the appropriate entry_body or accept_statement.
>
> (8) If A1 is part of the execution of an accept_statement or entry_body, and A2 is the action of returning from the corresponding entry call;
>
> (9) If A1 is part of the execution of a protected procedure body or entry_body for a given protected object, and A2 is part of a later execution of an entry_body for the same protected object;

(10)If A1 signals some action that in turn signals A2.

Erroneous Execution

(11) Given an action of assigning to an object, and an action of reading or updating a part of the same object (or of a neighboring object if the two are not independently addressable), then the execution of the actions is erroneous unless the actions are sequential. Two actions are sequential if one of the following is true:

(12) One action signals the other;

(13) Both actions occur as part of the execution of the same task;

(14) Both actions occur as part of protected actions on the same protected object, and at most one of the actions is part of a call on a protected function of the protected object.

A remote procedure call is a procedure call, so 9.10(3) implies that RPC's are signaling, so long as we view the call as taking place within the execution of a single task.

The only problem is that asynchronous RPC's are weird; the caller proceeds without awaiting return of the call. Thus, we need a special-case rule for that case.

As an example, suppose a task in one partition writes upon a shared variable in a shared passive partition. It may then do an RPC to notify other partitions that it has done writing. The other partitions may then safely read from that shared variable.

As a special case, consider a partition that initializes such a shared variable during that partition's elaboration. E.4(14) says:

> If a remote subprogram call is received by a called partition before the partition has completed its elaboration, the call is kept pending until the called partition completes its elaboration (unless the call is cancelled by the calling partition prior to that).

So other partitions may assume that the shared variable has been initialized, so long as they first do an RPC (that does not raise Communication_Error) to the initializing partition.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0087**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **E.5**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0087 The PCS may be defined by the user

Working Reference Number AI95-00082

Question

A(4) says:

> The implementation may restrict the replacement of language-defined compilation units. The implementation may restrict children of language-defined library units (other than Standard).

Is this intended to apply to the body of System.RPC, or its children? (No.)

May an implementation require that a particular version of System.RPC be used? (No.)

Summary of Response

An implementation that conforms to Annex E, and that supports pragma Remote_Call_Interface (which is not required -- see E.2.3(20)) must allow the user to compile a body for System.RPC, and to compile children of System.RPC.

Such an implementation must implement remote subprogram calls using (only) the facilities of System.RPC; the generated code is not allowed to depend on special properties of one particular implementation of System.RPC, but must work for any correct implementation of System.RPC.

Corrigendum Wording

Insert after E.5(24):

> The implementation of the RPC-receiver shall be reentrant, thereby allowing concurrent calls on it from the PCS to service concurrent remote subprogram calls into the partition.

the new paragraphs:

> An implementation shall not restrict the replacement of the body of System.RPC. An implementation shall not restrict children of System.RPC. The related implementation permissions in the introduction to Annex A do not apply.
>
> If the implementation of System.RPC is provided by the user, an implementation shall support remote subprogram calls as specified.

Discussion

The intent is that the PCS be implemented by the user, or by a third party vendor -- it need not be implemented by the Ada compiler vendor. Hence, it is important that the user be able to provide a body, and child units, for System.RPC. This requires:

> (1) The Ada compiler vendor must allow users to compile the body and children of System.RPC, despite A(4).
>
> (2) The Ada compiler must generate code that will work properly with any correct implementation of the PCS; thus, the generated code must use the defined interface, and only that interface, and not depend on details of a particular PCS implementation.

Thus, it would be correct for a validation test to provide a PCS implementation, and require the implementation to use that PCS in tests.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0088**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **F.3.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0088 Picture string grammar or composition rules need tightening

Working Reference Number AI95-00153

Question

The String "++++>" and like Strings with '>' unmatched by any '<' appear to be valid picture strings based on the following production sequence (from F.3.1):

picture string ::= ... | non_currency_picture_string

non_currency_picture_string ::= all_sign_number | ...

all_sign_number ::= all_sign_fore [...] [>]

all_sign_fore ::= sign_char {...} sign_char {sign_char | ...}

sign_char ::= + | - | <

Is the picture string "++++>" well formed? (No.)

Summary of Response

Picture strings such as "++++>" are not well formed.

Corrigendum Wording

Replace F.3.1(43):

- If a picture String has '+' or '-' as fixed_LHS_sign, in a floating_LHS_sign, or in an all_sign_number, then it has no RHS_sign.

by:

- If a picture String has '+' or '-' as fixed_LHS_sign, in a floating_LHS_sign, or in an all_sign_number, then it has no RHS_sign or '>' character.

Response

A '>' character can only appear in an all_sign_number if it contains '<' characters.

Discussion

The problem is that the production for all_sign_number does not use the RHS_sign production, it contains the literal '>'. The picture string grammar and composition constraints were folded into as few words as possible, too few in this case. Some of the composition constraints could have been included in a context-free grammar but at the expense of making it much longer, and more difficult to read.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0089**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **F.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0089 Incorrect picture string example

Working Reference Number AI95-00070

Question

The picture example in F.3.2(74) is invalid, since a floating currency symbol is not allowed in the same picture string as a zero suppression symbol.

Summary of Response

The example in F.3.2(74) has been corrected.

Corrigendum Wording

Replace F.3.2(74):

```
      123456.78     Picture:  "-$$$**_***_**9.99"
                    Result:   "bbb$***123,456.78"
                             "bbbFF***123.456,78" (currency = "FF",
                                                   separator = '.',
                                                   radix mark = ',')
```

by:

```
      123456.78     Picture:    "-$**_***_**9.99"
                    Result:     "b$***123,456.78"
                               "bFF***123.456,78" (currency = "FF",
                                                   separator = '.',
                                                   radix mark = ',')
```

Discussion

The example was incorrect.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0090**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Presentation**
REFERENCES IN DOCUMENT: **G.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0090 Should "pragma" be in boldface?

Working Reference Number AI95-00028

Question

Should "pragma" be in boldface? (Yes.)

Summary of Response

"pragma" should be in boldface.

Corrigendum Wording

Replace G.1.1(2):

```
generic
   type Real is digits <>;
package Ada.Numerics.Generic_Complex_Types is
   pragma Pure(Generic_Complex_Types);
```

by:

```
generic
   type Real is digits <>;
package Ada.Numerics.Generic_Complex_Types is
   pragma Pure(Generic_Complex_Types);
```

Discussion

This was an editing error.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0091**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **G.1.1**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0091 Polar implementation of complex exponentiation for negative exponents

Working Reference Number AI95-00156

Question

G.1.1(55) gives the following method for doing complex exponentiation in polar form:

> ... exponentiating the modulus by the given exponent; multiplying the argument by the given exponent, when the exponent is positive, or dividing the argument by the absolute value of the given exponent, when the exponent is negative; ...

The special case for the determining the argument of the result when the exponent is negative is incorrect. The method given for positive exponents should be applied for all exponents, including interestingly enough, zero exponents.

Summary of Response

The second sentence of G.1.1(55) should read:

> Implementations are also permitted to obtain the result of exponentiation of a complex operand, but not of a pure-imaginary operand, by converting the left operand to a polar representation, exponentiating the modulus by the given exponent, multiplying the argument by the given exponent, and reconverting to a Cartesian representation.

Corrigendum Wording

Replace G.1.1(55):

> Implementations may obtain the result of exponentiation of a complex or pure-imaginary operand by repeated complex multiplication, with arbitrary association of the factors and with a possible final complex reciprocation (when the exponent is negative). Implementations are also permitted to obtain the result of exponentiation of a complex operand, but not of a pure-imaginary operand, by converting the left operand to a polar representation; exponentiating the modulus by the given exponent; multiplying the argument by the given exponent, when the exponent is positive, or dividing the argument by the absolute value of the given exponent, when the exponent is negative; and reconverting to a cartesian representation. Because of this implementation freedom, no accuracy requirement is imposed on complex exponentiation (except for the prescribed results given above, which apply regardless of the implementation method chosen).

by:

> Implementations may obtain the result of exponentiation of a complex or pure-imaginary operand by repeated complex multiplication, with arbitrary association of the factors and with a possible final complex reciprocation (when the exponent is negative). Implementations are also permitted to obtain the result of exponentiation of a complex operand, but not of a pure-imaginary operand, by converting the left operand to a polar representation, exponentiating the modulus by the given exponent, multiplying the argument by the given exponent, and reconverting to a cartesian representation. Because of this implementation freedom, no accuracy requirement is imposed on complex exponentiation (except for the prescribed results given above, which apply regardless of the implementation method chosen).

Discussion

Here is a proof by example that the given method is incorrect:

Assume that the method described in the standard is correct.

Let a complex number X = i and let an integer n = -1.

Then X**n = 1/i = -i, argument(X) = pi/2 and n is negative.

So, according to G.1.1(55), argument(X**n) = (pi/2)/|-1| = pi/2, but argument(X**n) = argument(-i) = -pi/2.

Obviously, pi/2 is not equal to -pi/2 (even as an angle); i.e. a contradiction has been found. No zero-valued complex numbers were involved (they can mess things up). The only dubious assumption made was that the method described in G.1.1(55) was correct. So, it must not be.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0092**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Error**
REFERENCES IN DOCUMENT: **G.1.3**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0092 Does Complex_IO handle extended real literals?

Working Reference Number AI95-00029

Question

G.1.3(12) says that Complex_IO.Get reads a pair of optionally signed real literals. This is inconsistent with A.10.9(13-18), which allow certain extended forms of real literals in Float_IO.Get. Should Complex_IO.Get allow the same extended forms? (Yes.)

Summary of Response

The syntax of real literals read by Ada.Text_IO.Complex_IO.Get is the same as that of Ada.Text_IO.Float_IO.Get.

Corrigendum Wording

Replace G.1.3(12):

> The input sequence is a pair of optionally signed real literals representing the real and imaginary components of a complex value; optionally, the pair of components may be separated by a comma and/or surrounded by a pair of parentheses. Blanks are freely allowed before each of the components and before the parentheses and comma, if either is used. If the value of the parameter Width is zero, then

by:

> The input sequence is a pair of optionally signed real values representing the real and imaginary components of a complex value. These components have the format defined for the corresponding Get procedure of an instance of Text_IO.Float_IO (see A.10.9) for the base subtype of Complex_Types.Real. The pair of components may be separated by a comma or surrounded by a pair of parentheses or both. Blanks are freely allowed before each of the components and before the parentheses and comma, if either is used. If the value of the parameter Width is zero, then

Response

The syntax of real literals read by Ada.Text_IO.Complex_IO.Get is the same as that of Ada.Text_IO.Float_IO.Get. The same applies to Ada.Wide_Text_IO.Complex_IO.Get.

Discussion

The intent is that all input of real literals, as well as the 'Value attribute, accept the same syntax. This intent is reflected in AARM G.1.3(1.a), which suggests implementing Complex_IO in terms of Float_IO.

DEFECT REPORT

The submitter of a defect report shall complete the items in Part 2 and shall send the form to the Convener or the Secretariat of the WG with which the relevant editor's group is associated.

The WG Convener or Secretariat shall complete the items in Part 1 and circulate the defect report for review and response by the appropriate defect editing group.

The defect editor shall complete Part 3 and submit the completed report to the Convener or the Secretariat of the WG.

PART 1 - TO BE COMPLETED BY WG SECRETARIAT
DEFECT REPORT NUMBER: **8652/0093**
WG SECRETARIAT: **James W. Moore, Convener, ISO/IEC JTC 1/SC 22/WG 9**
DATE CIRCULATED BY WG SECRETARIAT: **2000/08/01**
DEADLINE ON RESPONSE FROM EDITOR: **2000/10/01**

PART 2 - TO BE COMPLETED BY SUBMITTER
SUBMITTER: **Erhard Ploedereder and Randall Brukardt, Ada Project Editors**
FOR REVIEW BY: **ISO/IEC JTC 1/SC 22/WG 9**
DEFECT REPORT CONCERNING: **ISO/IEC 8652:1995 Programming languages — Ada**
QUALIFIER: **Clarification required**
REFERENCES IN DOCUMENT: **H.3.2**
NATURE OF DEFECT (complete, concise explanation of the perceived problem): **See Question on next page.**
SOLUTION PROPOSED BY THE SUBMITTER (optional): **See Summary of Response on next page.**

PART 3 - EDITOR'S RESPONSE
ANY MATERIAL PROPOSED FOR PROCESSING AS A TECHNICAL CORRIGENDUM TO, AN AMENDMENT TO, OR A COMMENTARY ON THE INTERNATIONAL STANDARD OR DIS FINAL TEXT IS ATTACHED TO THIS COMPLETED REPORT: **See next page.**

8652/0093 Only the current unit is affected by pragma Inspection_Point

Working Reference Number AI95-00207

Question

In the standard, pragma Inspection_Point is not a configuration pragma. However, consider the following example:

```
procedure P is
   A : Integer := 1;
begin
   Q;
   -- A is not used after this point
end;

procedure Q is
begin
   ...
   pragma Inspection_Point;
end;
```

In our example A must be inspectable at the inspection point, according to H.3.2(5).

Must we be able to find the value of A, even if it is in a different compilation unit? (No.) If so, the dead value must be stored before calling Q. Or does pragma Inspection_Point apply only to objects in the current compilation unit? (No.)

Summary of Response

Pragma Inspection_Point applies to all variables given as arguments or, if there are no arguments, to all variables visible at the inspection point.

Corrigendum Wording

Replace H.3.2(5):

> An *inspection point* is a point in the object code corresponding to the occurrence of a pragma Inspection_Point in the compilation unit. An object is *inspectable* at an inspection point if the corresponding pragma Inspection_Point either has an argument denoting that object, or has no arguments.

by:

> An *inspection point* is a point in the object code corresponding to the occurrence of a pragma Inspection_Point in the compilation unit. An object is *inspectable* at an inspection point if the corresponding pragma Inspection_Point either has an argument denoting that object, or has no arguments and the object is visible at the inspection point.

Response

A pragma Inspection_Point without parameters applies only to all variables visible at the point.

Discussion

A pragma Inspection_Point with arguments requires that all the objects listed be visible. It was the intent that a pragma Inspection_Point without arguments be a convenient shorthand for listing all objects which could have been given as arguments.

It follows that a pragma Inspection_Point without arguments applies to all those objects visible at that point.

In the example the object A is not visible at the place of the pragma and therefore its value need not be available.

The pragma does not apply just to objects in the current compilation unit since global objects in another compilation unit might be visible. Thus consider

```
package P is
   A: Integer;
private
   B: Integer;
end;

with P;
procedure Q is
begin
   pragma Inspection_Point;
   ...
end Q;
```

Since A is visible at the place of the pragma its value must be available for inspection. The same does not apply to B.

Notwithstanding the above, any compiler conforming to Annex H might have a mode of operation that enables all global variables (visible or not) to be inspected at any point.