INTERNATIONAL STANDARD

ISO/IEC 10514-2

First edition
1998-12-01

Information technology — Programming languages —

Part 2:
Generics in Modula-2

Technologies de l'information — Langages de programmation —
Partie 2: Éléments génériques en modula 2

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

Reference number
ISO/IEC 10514-2:1998(E)

Contents

Page

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

© ISO/IEC 1998

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and micro-film, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

Foreword

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

In the field of information technology, ISO and IEC have established a joint

Introduction

This part of ISO/IEC 10514 specifies the form and meaning of programs written in ISO Standard Modula-2 with Generic extensions and by reference to that specification lays down requirements for implementations of ISO Standard Modula-2 with Generic extensions.

The reader is referred to International Standard ISO/IEC 10514-1 (herein referred to as "the Base Language") for introductory remarks on the programming language Modula-2.

This part of ISO/IEC 10514 defines ISO Standard Modula-2 with Generic extensions by additions to the Base Language without changing the meaning of any parts of the Base Language.

This part of ISO/IEC 10514 does not provide a formal specification of ISO Standard Modula-2 with Generic extensions, although it is the intention of WG13 to construct the appropriate VDM-SL descriptions for the syntax and semantics described herein when committee resources permit.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

Information technology — Programming languages —

Part 2:
Generics in Modula-2

1 Scope

1.1 General

This part of ISO/IEC 10514 specifies extensions to allow generic programming facilities to be added to the base Modula-2 language defined in International Standard ISO/IEC 10514-1 without altering the meaning of valid programs allowed by the Base Language (except for the use of the new keyword introduced by this standard—see clause 5).

1.2 Specifications included in this part of ISO/IEC 10514

iTeh STANDARD PREVIEW
(standards.iteh.ai)

In addition to the specifications included in the Base Language this part of ISO/IEC 10514 provides specifications for:

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

— required symbols for programs written in ISO Standard Modula-2 with Generic extensions;
— the lexical structure and semantics of programs written in ISO Standard Modula-2 with Generic extensions;
— the syntax of programs written in ISO Standard Modula-2 with Generic extensions;
— violations of the rules for the use of the Generic extensions that a conforming implementation is required to detect;
— further compliance requirements for implementations, including documentation requirements.

1.3 Relationship to ISO/IEC 10514-1

This part of ISO/IEC 10514 is part two of the multi-part Standard ISO/IEC 10514. This part of ISO/IEC 10514 extends and modifies the Base Language ISO/IEC 10514-1, but the adoption of this part of ISO/IEC 10514 is optional with respect to the Base Language. This part of ISO/IEC 10514 is also independent of any other parts of ISO/IEC 10514, except for part 1, and can be adopted either together with or independently of such other parts.

1.4 Specifications not within the scope of this part of ISO/IEC 10514

In addition to the categories of specifications excluded by the Base Language this part of ISO/IEC 10514 provides no specifications for:

— the method by which specific refinements are constructed from generic library modules;
— the method by which generic library modules, their associated refining modules, and the refinements produced by these are stored (including any correspondence between the module names and system file names where files are used).

2 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10514. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 10514 are encouraged to investigate the possibility of applying

the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 10514-1:1996, *Information technology — Programming languages — Part 1: Modula-2, Base Language.*

3 Definitions, Structure and Conventions

3.1 Definitions

For the purposes of this part of ISO/IEC 10514, the definitions given in ISO/IEC 10514-1 and the following definitions apply.

3.1.1 Generic separate module

A new kind of separate module having formal parameters that can be either type parameters and/or constant value parameters. A generic separate module serves as a template for constructing specific refinements of itself that have been customized using actual types and/or actual constant expressions.

NOTE 1 — A generic separate module consists of a generic definition module and a generic implementation module. The generic definition module is a template from which a definition module can be refined. The generic implementation module is a template from which an implementation module can be refined.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

3.1.2 Refiner or Refining module

A new kind of module that is a means of supplying actual parameters for the purpose of creating a refinement of the generic separate module from which the refinement is being made.

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

NOTE 2 — A refining module can be a separate module, in which case its definition module produces a refinement of the Generic Modula-2 definition module of the generic separate module from which the refinement is being made. The implementation module of such a refining separate module refines the implementation module of the generic separate module from which the refinement is being made.

NOTE 3 — A refining module can be a local module, in which case its refinement has as its qualified export list the items defined in the generic definition module of which it is a refining module. Such a refining local module refines the implementation module of the generic separate module from which the refinement is being made.

3.1.3 Refinement

An abstract entity (either a whole module or an item it contains) constructed from a generic separate module by specifying in a refining module the name of a generic separate module to be refined from and actual types and/or actual constant expressions to evaluate and then substitute for the formal parameters of the generic separate module.

NOTE 4 — The refining module is not the same entity as the refinement produced from it.

3.1.4 Refine

The act of constructing a refinement from a generic separate module by using a refining module.

NOTE 5 — Abstractly, the refining of a generic separate module to a specific refinement is the task of the translator. However, nothing in this part of the multi-part standard precludes an implementation separating this particular translation task from others. For instance, refinement could be performed separately before other translation tasks.

NOTE 6 — The effect of refinement is the same as if a refinement of the generic separate module had been written directly in the base language with the formal parameters replaced by the results of evaluating the actual parameters. Depending on the implementation strategy, this might not be quite the same as saying that refinement results in a program in the base language (except in the abstract sense), for there is no requirement in this part of ISO/IEC 10514 for a specific intermediate form in which a refinement finds some expression as a file.

TERMINOLOGY NOTE —WG13 is already using the term "instantiate" for use in Object Oriented Modula-2, and has chosen to use "refine" in ISO Standard Modula-2 with Generic extensions.

3.1.5 Generic

A property of both the whole and of any item defined within a generic separate module, regardless of whether the item itself contains or needs module parameters.

3.2 Structure of the Formal Definition

This part of the multi-part International Standard states its requirements in the same form as the Base Language with the exception that it does not include formal expression of semantics in VDM-SL at this time.

3.3 Conventions

The conventions used in this part of ISO/IEC 10514 are to be interpreted in the same way as in the Base Language with the exception that this part of ISO/IEC 10514 does not include VDM-SL at this time.

4 Requirements for Implementations

iTeh STANDARD PREVIEW
(standards.iteh.ai)

4.1 General Requirements

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

A conforming implementation of ISO Standard Modula-2 with Generic extensions meets the requirements for Modula-2 implementations that are laid down in the Base Language. In addition, it meets the requirements of this clause:

4.2 Translation

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall accept compilation modules for translation from source code when they contain the additional lexical form defined in clause 5, and when this is n the syntact ic form specified in clause 6. It shall also accept the lexical forms defined in the Base Language when they are used in the (new) syntactic forms specified in ISO Standard Modula-2 with Generic extensions.

NOTE — The effect of accepting for translation the new compilation modules is discussed in the appropriate subclauses of clause 6 of this part of the multi-part standard.

4.3 Source Code Representation

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall provide the additional keyword specified in clause 5 and shall recognize keywords and identifiers as specified in that clause, including those situations where keywords and symbols from the base language are used in new syntactic constructs in this part of ISO/IEC 10514.

4.4 Ordering of Declarations

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation.

4.5 Predefined Entities

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation.

4.6 Library Modules

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation. No new library modules are specified by this part of ISO/IEC 10514.

NOTE — This does not preclude the possibility that later editions of or amendments to this part of the multi-part standard might include such modules.

4.7 Errors

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation, with the following changes:

— It shall detect any new errors defined in this part of ISO/IEC 10514 in a manner consistent with the detection and reporting of errors required by the Base Language.
— In standard mode a conforming implementation of ISO Standard Modula-2 with Generic extensions shall treat the use of extensions that are not specified by this part of ISO/IEC 10514 or by another standard extension of the Base Language as errors. Conformance to standards parallel to this one (if any) is on an additive basis, so that two or more such standard extensions can be conformed to simultaneously.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

NOTE — The intent of this provision is to allow a version of Modula-2 to support, for example, both genericity and object orientation (see ISO 10514-3).

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

4.8 Exceptions

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation. The rules shall be applied after the construction of any refinements.

NOTE 1— This means, for instance, that two refinements of a single generic separate module constitute two different sources for exceptions.

Because the process of refinement logically takes place before the translation of the refinement, the only new errors defined in this part of ISO/IEC 10514 are syntactical and shall be detected by the translator. No new exceptions are therefore defined.

NOTE 2 — This does not preclude the possibility that later editions of or amendments to this part of the multi-part standard might include library modules that define exceptions. Nor does it prohibit implementations from providing library modules with their own exceptions.

4.9 Implementation-dependencies

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation.

4.10 Documentation

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation.

4.11 Statement of Compliance

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation and in addition, a separate compliance statement shall be made citing the degree of compliance with this part of ISO/IEC 10514.

4.12 Minimum requirements

A conforming implementation of ISO Standard Modula-2 with Generic extensions shall have rules identical in this respect to a conforming Modula-2 implementation.

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

5 The Lexis

5.1 Relationship to the Base Language

The lexis of ISO Standard Modula-2 with Generic extensions is based directly on the lexis of the base language as defined in ISO/IEC 10514-1. All syntax elements of the base language keep their representation and all elements of the base language retain their same semantics; a new keyword that is needed to express the new language elements is defined in this clause.

NOTE - A conforming program written in the base language (or in the base language augmented by some other standard extension parallel to this part of the multi-part standard) is translated correctly by a translator conforming to ISO Standard Modula-2 with Generic extensions except in the case that the keyword defined in this part of the multi-part standard is used as an identifier in the program.

5.2 Keyword

Only one new keyword is defined; it is used to denote generic separate modules.

Concrete Syntax

keyword = base language keyword | "GENERIC" ;

NOTE — ISO Standard Modula-2 with Generic extensions does not define any new pervasive identifiers or any other new lexis elements beyond those already present in the base language.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

6 The Language

6.1 The Model

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

ISO Standard Modula-2 with Generic extensions is an extension of the base language to provide facilities for generic programming. The provided model is characterized as follows:
— Genericity is defined at the separate module level only. (All the contents of a generic separate module are regarded as generic, but no item within a module can be declared as generic independently of the module containing it).
— Generic separate modules have formal parameters that are either type parameters or constant value parameters.
— Refining separate modules have actual parameters that are compatible with the formal parameters of the generic separate module from which the refinement is being made.
— Refining local modules have actual parameters that are compatible with the formal parameters of the generic separate module from which the refinement is being made.
— Formal/actual module parameter correspondence semantics are identical to those employed in the base language for constant value parameters of procedures. (There are no variable parameters allowed in generic separate module parameter lists; neither can variables be used to pass values to constant value parameters in generic separate module parameter lists).
— Refinement takes place not later than translation time.

ISO Standard Modula-2 with Generic extensions incorporates all the syntax and semantics of the base language without alteration. This clause indicates only the additions (new syntax and semantics) required to support generic programming.

6.2 Programs, Program Modules, and Separate Modules

6.2.1 The New Modules

Generic separate modules are extensions of the separate modules defined in the Base Language. Each has a generic definition module that shall exist in order to check the well-formedness of any other module that depends on it. A generic separate module also has a generic implementation module. Another module does not import items from a generic separate module; rather, it imports them from a refinement of a generic separate module.

Refining separate modules are variations of the separate modules defined in the Base Language. Each has a refining definition module that shall exist in order to check the well-formedness of any other module that depends on it. A refining separate module also has a refining implementation module. Each refining separate module depends upon some generic separate module, and this imposes the constraint that both the refining separate module and the generic separate module from which it refines shall exist before the well-formedness of the refinement produced by a refining separate module can be checked.

The fact that the translation of a program depends upon the prior translation of the modules it imports means that the refinement shall exist in order to be translated in the correct order. Thus, the process of refinement logically precedes that of the establishment of program compilation dependencies, which in turn logically precedes that of the translation of the program. However, the physical means by which this is actually achieved is implementation defined. Thus, implementations are free to separate the step of refinement from that of translation, or to combine the two as they see fit. This also means that, although a refinement of a generic separate module can be thought of in a logical sense as itself being a module (separate or local), there is no need to define its syntax or semantics as such, for these derive from the generic separate module and the refiner creating the refinement from it and in turn are correct (or not) according to the rules of the Base Language.

NOTE — A consequence of this is that in systems that employ files, a refinement might find physical expression as a file independent of the file(s) containing the refining module. However, this is not required, and if the implementation strategy employed is to have the translator construct the refinement as part of the translation process, no such independent files need be created.

6.2.2 Programs and Compilation Modules

Four new kinds of compilation modules are added to the list provided in the Base Language. This produces the following change:

Concrete Syntax

iTeh STANDARD PREVIEW
(standards.iteh.ai)

compilation module = program module | definition module | implementation module | generic definition module | generic implementation module | refining definition module | refining implementation module ;

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

6.2.3 Generic Definition Module

A generic definition module is an extension of the definition module in the Base Language, and the rules for definition modules given in the Base Language apply to generic definition modules, with the following additions.

Concrete Syntax

generic definition module =
"GENERIC", "DEFINITION", "MODULE", module identifier, [formal module parameters], semicolon,
import lists, definitions,
"END", module identifier, period ;

Semantics

The effect of processing a generic definition module by a translator is implementation defined, but
— Any identifiers it imports shall be distinct from each another and from any identifiers it defines.
— Any identifiers it defines shall be distinct from each other and from any identifiers it imports.
— Its own name is added to the environment. This allows other modules to import it.
— The identifiers it defines are NOT added to the environment. (When the module is later refined, the refinements of those identifiers will be added to the environment instead of the identifiers defined by the generic separate module). This means that the translator shall report as an error any attempt to perform unqualified import of an identifier defined in a generic definition module or to employ such an identifier when qualified by the name of a generic definition module.
— Apart from its own name being added to the environment, no translation of a generic definition module takes place until a refining definition module is translated, and at that time the effect of the translation depends on the actual parameters supplied by the refining module.

NOTE 1— The effect of this part of the multi-part standard is to require that refinements be checked in the same way as any base language module. These rules do not either require or preclude an implementation having the translator perform such consistency checks as might be possible on the definition module of a generic separate module when presenting it to the translator.

NOTE 2— The entities defined in a generic separate module are not available for use by other modules until they have been refined by the translation of a refining module that refines from the generic separate module in question. (They can of course be used within the generic separate module itself). It is, therefore, an error that the translator shall report, to attempt to employ in another body items from a generic separate module rather than from a refinement of one. This does not of course preclude any module (including a generic one) from importing a generic separate module for the purpose of refining it.

Examples

Conventional data structures such as lists, queues, and stacks can be constructed generically, and parameterized to provide them with sufficient information about the data expected to be entered into the structure.

NOTE 3 — Several of these examples are dealt with further in the clauses on generic implementations and refining definitions and implementations.

Example 1: The first example is a generic definition of a data structure. In such applications, it is expected that the structure will be manipulated independent of the kind of element it contains. All the work of programming is in the structure manipulation; the provision of the items to enter into it is a refining detail.

```
GENERIC DEFINITION MODULE Stacks (Element: TYPE);

CONST
  StackSize = 100;

PROCEDURE Push (item : Element);

PROCEDURE Pop (VAR item : Element);

PROCEDURE Empty () : BOOLEAN;

END Stacks.
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

NOTE 4— Specification of a constant such as *StackSize* in the manner of this example constrains all the refinements of this module. If that were not the intention, such an item could instead be made a module parameter and then specified by the refinements.

NOTE 5 — Such a constant can be used in the corresponding generic implementation module. However, depending on the implementation strategy chosen, the correctness of such use might not be checked until the refinement of that implementation is performed.

Example 2: Sometimes the data structure itself needs parameterization.

```
GENERIC DEFINITION MODULE Matrix (Rows, Cols : CARDINAL; MatrixElement : TYPE);

TYPE
  TMatrix = ARRAY [0 .. Rows-1]
       OF ARRAY [0 .. Cols-1] OF MatrixElement;

PROCEDURE Invert (VAR m : TMatrix);

END Matrix.
```

Example 3: This illustration shows a generic technique, as opposed to a generic Abstract Data Type (ADT).

```
GENERIC DEFINITION MODULE Validate (PType : TYPE; PValidProc : ValidProcType);

TYPE
  ValidProcType = PROCEDURE (item : PType) : BOOLEAN;
   (* Note the forward reference in the module parameter list *)

PROCEDURE Valid (item : PType) : BOOLEAN;

END Validate.
```

Example 4: (An outline of a generic sort) This example illustrates the abstraction of a generic technique applied to an existing kind of structure (an array), rather than to a user-defined structure.

```
GENERIC DEFINITION MODULE Sorts (Item : TYPE; GenCompare : CompareProc);

FROM Comparisons IMPORT
  CompareResults;

TYPE
  CompareProc = PROCEDURE (Item, Item) : CompareResults;

PROCEDURE Quick (VAR data : ARRAY OF Item);

  (* Other procedures and functions could be included as well. *)
END Sorts.
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)

NOTE 6 — The type identifiers of formal parameters can forward reference types defined in the definition module of the generic separate module itself.

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

Example 5: (Parameters are optional) This example illustrates that although in the majority of cases generic separate modules will be parameterized, this is not required, and non-parameterized instances might also be useful. This module defines a counter, any number of instances of which can be refined under different names.

```
GENERIC DEFINITION MODULE Counter;

PROCEDURE Inc;

PROCEDURE Reset;

PROCEDURE Count () : CARDINAL;

END Counter.
```

6.2.4 Generic Implementation Module

A generic implementation module is an extension of the implementation module in the sense of the Base Language, and the rules for implementation modules (and their relationships with their definition modules) given in the Base Language apply, with the following additions.

Concrete Syntax

generic implementation module =
"GENERIC", "IMPLEMENTATION", "MODULE", module identifier, [interrupt protection], [formal module parameters], semicolon,
import lists, module block,
module identifier, period ;

Semantics

The effect of translating the implementation module of a generic separate module by a translator is implementation defined, but:
— The definition module of the generic separate module shall already exist and have both the same name and the same parameters.
— Any identifiers that the implementation module of a generic separate module imports shall be distinct from each other, from any identifiers it defines, and from any identifiers defined in its corresponding generic definition module.
— Any identifiers that the implementation module of a generic separate module defines shall be distinct from each other, from any identifiers it imports, and from any identifiers defined in its corresponding generic definition module.
— Other than possibly checking such consistencies, the translation of a generic implementation module cannot be performed until it is refined by a refining module, and at that time the effect of the translation depends on the actual parameters supplied by the refining module.
— When a refinement of a generic implementation module is translated, the effect is the same as translating a copy of the generic implementation module with the results of evaluating the actual parameters supplied in the refinement substituted for the formal parameters in the generic separate module, and at that time, only the rules of the base language need be applied to perform the translation.

iTeh STANDARD PREVIEW
(standards.iteh.ai)

NOTE 1 — The effect of this part of the multi-part standard is to require that refinements be checked in the same way as any base language module. These rules neither require nor preclude an implementation having the translator perform such consistency checks as might be possible on the implementation module of a generic separate module by presenting it to the translator separately.

ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-

The parameters of the generic implementation module shall be identical to the parameters of the corresponding generic definition module.

NOTE 2 — The interrupt protection on the generic implementation module (when provided) applies to all refinements of this module, whether separate or local, and such refinements can not have a protection expression of their own.

If a programmer employs code in the generic implementation that assumes something about a type that is passed to it via one of the formal parameters, and then a refinement is done passing an actual type inconsistent with this assumption, the error shall be detected upon attempting to translate the refinement of the implementation (i.e. after the refinement has been constructed by the refiner). For instance, suppose a generic implementation used the operator "<" rather than having a compare procedure passed to it as a parameter (as done in example 4 in 6.2.3 and 6.2.4. The genericity is then restricted to data types that could use "<" and if some other data type were employed, the translator is required by the base language standard to report the error when attempting to translate the refinement of the implementation module. Without imposing complicated syntax on the form of generic separate modules to specifically delimit or control the use of built-in operations and procedures, there is no solution to this difficulty other than ensuring that the code for generic separate implementation modules is itself written in as "generic" a fashion as will permit the envisioned refinements of it to be correct.

NOTE 3 — A consequence of such application of the rules of the Base Language to the translation of refinements (i.e. after the refinement is done) is that a generic separate module that is syntactically correct and has been refined by a syntactically correct refining module can still result in a refined module that is not correct.

Examples

Example 1: What follows is a possible generic implementation of the first example in 6.2.3:

```
GENERIC IMPLEMENTATION MODULE Stacks (Element : TYPE);

VAR
  stack    :  ARRAY [0..StackSize] OF Element;
   stackPtr : CARDINAL;
(* One could also arrange for StackSize to be a parameter. *)

PROCEDURE Push (item : Element);
BEGIN
 stack[stackPtr] := item;
  INC (stackPtr);
END Push;

PROCEDURE Pop (VAR item : Element);
BEGIN
  DEC (stackPtr);
  item := stack[stackPtr];
END Pop;

PROCEDURE Empty () : BOOLEAN;
BEGIN
  RETURN stackPtr = 0
END Empty;

BEGIN (* module body initialization *)
  stackPtr := 0
END Stacks.
```

iTeh STANDARD PREVIEW
(standards.iteh.ai)
ISO/IEC 10514-2:1998
https://standards.iteh.ai/catalog/standards/sist/e72dbb55-c4c1-476e-8817-6dcdea76e289/iso-iec-10514-2-1998

NOTE 4 — The constant *StackSize* is from the corresponding definition module. It can be used in this generic implementation module. However, depending on the implementation strategy chosen, the correctness of such use might not be checked until the refinement of that implementation is performed.

Example 2: The next module is a corresponding implementation for the generic definition of example 2 in 6.2.3.

```
GENERIC IMPLEMENTATION MODULE Matrix (Rows, Cols : CARDINAL; MatrixElement : TYPE);

(* Note that, as in any implementation module, the generic implementation module has access to the <generic> types defined
in the corresponding definition. *)

PROCEDURE Invert (VAR m : TMatrix);
BEGIN
  (* your favourite technique *)
END Invert;

END Matrix.
```

Example 3: The next module is a corresponding implementation for the generic definition of example 3 in 6.2.3.

```
GENERIC IMPLEMENTATION MODULE Validate (PType : TYPE; PValidProc : ValidProcType);

PROCEDURE Valid (item : PType) : BOOLEAN;
BEGIN
```