ISO/IEC JTC1/SC22/WG9 N397, 5 October 2001

Initial Working Draft of Revised ISO/IEC 13813

Information technology — Programming languages — Generic packages of real and complex array declarations for Ada

PROJECT: JTC1.22.10.02

SOURCE: ISO/IEC JTC1/SC22/WG9

DATE: October 1, 2001

STATUS: IS published; revisions pending

EDITOR: Donald W. Sando
The Boeing Company
P.O. Box 3707, MS 4A-25
Seattle WA 98124-2207
USA
email: donald.w.sando@boeing.com

This version of the International Standard is almost, but not quite, identical to the version printed by ISO. The textual content is the same. It is marked as a draft and has different page dimensions. Because the text has been prepared to format properly in the A4 format required by ISO, some page breaks may occur in awkward places in this version.

Contents

Annexes

Foreword

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

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 13813 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

Annexes A and B form an integral part of this International Standard. Annexes C and D are for information only.

Introduction

The generic packages described here are intended to provide the basic real and complex vector and matrix operations from which portable, reusable applications can be built. This International Standard serves a broad class of applications with reasonable ease of use, while demanding implementations that are of high quality, capable of validation and also practical given the state of the art.

The specifications included in this International Standard are presented as compilable Ada specifications in annexes A and B with explanatory text in numbered sections in the main body of text. The explanatory text is normative, with the exception of notes (labeled as such).

The word "may," as used in this International Standard, consistently means "is allowed to" (or "are allowed to"). It is used only to express permission, as in the commonly occurring phrase "an implementation may"; other words (such as "can," "could" or "might") are used to express ability, possibility, capacity or consequentiality.

Information technology — Programming languages — Generic packages of real and complex array declarations for Ada

1 Scope

This International Standard defines the specifications of two generic packages of vector and matrix operations called `Generic_Real_Arrays` and `Generic_Complex_Arrays`. The specifications of non-generic packages called `Real_Arrays` and `Complex_Arrays` are also defined, together with those of analogous packages for other precisions. This International Standard does not provide the bodies of these packages.

This International Standard specifies certain fundamental vector and matrix arithmetic operations for real and complex numbers. They were chosen because of their utility in various application areas.

This International Standard is applicable to programming environments conforming to ISO/IEC 8652.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 8652, *Information technology — Programming languages — Ada.*

3 Types and operations provided

The following four array types are exported by the packages provided by this International Standard:

```
Real_Vector             Real_Matrix
Complex_Vector          Complex_Matrix
```

Two composite types with elements of type `Real` are provided, `Real_Vector` and `Real_Matrix`, to represent real vectors and matrices; and two composite types with elements of type `Complex` are provided, `Complex_Vector` and `Complex_Matrix`, to represent complex vectors and matrices.

The following eigtheen operations are provided:

```
"+"                     "-"                     "*"                     "/"
"**"                    "abs"                   Conjugate               Transpose
Re                      Im                      Set_Re                  Set_Im
Compose_From_Cartesian  Modulus                 Argument                Compose_From_Polar
Unit_Vector             Identity_Matrix
```

These are the usual mathematical operators (`+`, `-`, `*` and `/`) for real and complex vectors and matrices (together with analogous componentwise operations for vectors); the exponentiation operator (`**`) for real and complex vectors; the absolute value operator (`abs`) for real and complex vectors and matrices; the conjugate operation (`Conjugate`) for complex vectors and matrices; the transpose operation (`Transpose`) for real and complex matrices; the cartesian component operations (`Re`, `Im`, `Set_Re`, `Set_Im` and `Compose_From_Cartesian`) for complex vectors and matrices, for selecting components and for composing from components; the polar component operations (`Modulus`, `Argument` and `Compose_From_Polar`) for complex vectors and matrices, for selecting components and for composing from components; and the initializing operations (`Unit_Vector` and `Identity_Matrix`) for real and complex vectors and matrices.

4 Instantiations

This International Standard describes generic packages `Generic_Real_Arrays` and `Generic_Complex_Arrays`. Each package has a generic formal parameter, which is a generic formal floating-point type named `Real`. At instantiation, this parameter determines the precision of the arithmetic.

This International Standard also describes non-generic packages `Real_Arrays` and `Complex_Arrays`, which provide the same capability as instantiations of the packages `Generic_Real_Arrays` and `Generic_Complex_Arrays`. It is required that non-generic packages be constructed for each precision of floating-point type defined in package `Standard`.

5 Implementations

An implementation of the array operations defined in `Generic_Real_Arrays` and `Generic_Complex_Arrays` shall conform to all of the implementation requirements specified for the corresponding (scalar) real type operations in ISO/IEC 8652. An implementation of the array operations defined in `Generic_Complex_Arrays` shall also conform to all of the implementation requirements specified for the corresponding (scalar) complex type operations in ISO/IEC 8652.

The accuracy requirements for the results of array operations are defined in terms of corresponding accuracy requirements, specified in ISO/IEC 8652, on their (real or complex) scalar elements, unless the mathematical definition of the operation includes an inner product (indicated in the specifications as such). The accuracy of operations involving inner products is beyond the scope of this International Standard, except that an implementation shall document what, if any, extended-precision accumulation of intermediate results is used to implement such inner products.

Implementations of `Generic_Complex_Arrays` shall provide both a strict mode in which the accuracy requirements are observed, and an opposing relaxed mode, as defined in the Numerics Annex of ISO/IEC 8652. Either mode may be the defualt mode, and the two modes need not actually be different. This is consistent with the numeric performance requirements for complex scalar arithmetic, and may in fact be inherited from an implementation of the package `Ada.Numerics.Generic_Complex_Types` specified in ISO/IEC 8652.

Implementations are allowed to make reasonable assumptions about the environment in which they are to be used, but only when necessary in order to match algorithms to hardware characteristics in an economical manner. For example, an implementation is allowed to limit the precision it supports (by stating an assumed maximum value for `System.Max_Digits`), since portable implementations would not, in general, be possible otherwise. All such limits and assumptions shall be clearly documented. By convention, an implementation of `Generic_Real_Arrays` and `Generic_Complex_Arrays` is said not to conform to this International Standard in any environment in which its limits or assumptions are not satisfied, and this International Standard does not define its behavior in that environment. In effect, this convention delimits the portability of implementations.

In implementations of `Generic_Complex_Arrays`, all operations involving mixed real and complex arithmetic are required to construct the result by using real arithmetic (instead of by converting real values to complex values and then using complex arithmetic). This facilitates support for a future Ada binding to IEC 559:1989.

Some hardware and their accompanying Ada implementations have the capability of representing and discriminating between positively and negatively signed zeros as a means (for example) of preserving the sign of an infinitesimal quantity that has underflowed to zero. Implementations in which `Real'Signed_Zeros` is `True` should attempt to provide a rational treatment of the signs of zero results, result components and scalar elements of composite results.

6 Exceptions

The `Constraint_Error` exception, declared in package `Standard` of ISO/IEC 8652, is raised by a subprogram in these generic packages when the argument(s) of the subprogram violate one or more of the conditions for matching elements of arrays (as in predefined equality); that is, for dyadic array operations, the bounds of the given left and right array operands need not be equal, but their appropriate vector lengths or row and/or column lengths (for matrices) shall be equal.

The `Argument_Error` exception, declared in package `Ada.Numerics` of ISO/IEC 8652, is raised by a subprogram in `Generic_Complex_Arrays` when the argument(s) of the subprogram violate one or more of the conditions given in the subprogram's definition.

NOTES

1 These conditions are related only to the mathematical definition of the subprogram and are therefore implementation independent.

2 These conditions are inherited from the corresponding scalar subprogram defined in `Ada.Numerics.Generic_Complex_Types` of ISO/IEC 8652.

An implementation shall raise the `Constraint_Error` exception for signaling division by zero in the following specific cases where the corresponding mathematical results, or components thereof, are infinite, provided `Real'Machine_Overflows` is `True`:

a) array operations whose mathematical definition involves division of an element by (real or complex) zero;

b) array operations whose mathematical definition involves exponentiation of (real or complex) zero by a negative (integer) exponent;

If `Real'Machine_Overflows` is `False`, the result for each of the foregoing specific cases is unspecified.

The `Constraint_Error` exception shall also be raised by a subprogram for all of the exceptional conditions related to real and complex types as defined in ISO/IEC 8652, provided `Real'Machine_Overflows` is `True`.

For the case of floating-point overflow, some of the operations are allowed to raise `Constraint_Error` for certain arguments for which neither component of the result can overflow, provided `Real'Machine_Overflows` is `True`. This freedom is granted for operations involving either an inner product or complex exponentiation. Permission to signal overflow in these cases recognizes the difficulty of avoiding overflow in the computation of intermediate results, given the current state of the art.

Besides `Argument_Error` and `Constraint_Error`, the only exceptions allowed during a call to a subprogram in these packages are the other predefined exceptions declared in package `Standard` of ISO/IEC 8652.

7 Generic Real Arrays Package

The generic package `Generic_Real_Arrays` defines operations and types for real vector and matrix arithmetic. One generic formal parameter, the floating-point type `Real`, is defined for `Generic_Real_Arrays`. The corresponding generic actual parameter determines the precision of the arithmetic to be used in an instantiation of this generic package.

The Ada package specification for `Generic_Real_Arrays` is given in annex A.

7.1 Types

Two types are defined and exported by `Generic_Real_Arrays`. The composite type `Real_Vector` is provided to represent a vector with elements of type `Real`; it is defined as an unconstrained, one-dimensional array with an index of type `Integer`. The composite type `Real_Matrix` is provided to represent a matrix with elements of type `Real`; it is defined as an unconstrained, two-dimensional array with indices of type `Integer`.

7.2 `Real_Vector` arithmetic operations

```
function "+" (Right : Real_Vector) return Real_Vector;
function "-" (Right : Real_Vector) return Real_Vector;
function "abs" (Right : Real_Vector) return Real_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of `Right`. This is also the standard mathematical operation for vector identity, negation and absolute value.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

```
function "+" (Left, Right : Real_Vector) return Real_Vector;
function "-" (Left, Right : Real_Vector) return Real_Vector;
function "*" (Left, Right : Real_Vector) return Real_Vector;
function "/" (Left, Right : Real_Vector) return Real_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of `Left` and the matching element of `Right`. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is `Left'Range`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

```
function "**" (Left  : Real_Vector;
               Right : Integer) return Real_Vector;
```

This operation returns the result of applying the standard mathematical operation for exponentiation by an integer power to each element of `Left`. The index range of the result is `Left'Range`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised if for some integer `I` (in the index range of `Left`), `Left(I)` $= 0.0$ and `Right` < 0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of exponentiation by an integer power, as defined in ISO/IEC 8652.

```
function "*" (Left, Right : Real_Vector) return Real'Base;
```

This operation returns the inner (dot) product of `Left` and `Right`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

7.3 `Real_Vector` scaling operations

```
function "*" (Left  : Real'Base;
              Right : Real_Vector) return Real_Vector;
```

This operation applies the standard mathematical operation for scaling a vector `Right` by a real number `Left`. The index range of the vector result is `Right'Range`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652.

```
function "*" (Left  : Real_Vector;
              Right : Real'Base) return Real_Vector;
function "/" (Left  : Real_Vector;
              Right : Real'Base) return Real_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector `Left` by a real number `Right`. The index range of the vector result is `Left'Range`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

7.4 Other `Real_Vector` operations

```
function Unit_Vector (Index : Integer;
                      Order : Positive;
                      First : Integer := 1) return Real_Vector;
```

This function returns a "unit vector" with `Order` elements and a lower bound of `First`. All elements are set to 0.0 except for the `Index` element which is set to 1.0. The exception `Constraint_Error` is raised if `Index` $<$ `First`, `Index` $>$ `First` + `Order` $-$ 1 or if `First` + `Order` $-$ 1 $>$ `Integer'Last`.

This function is exact.

7.5 `Real_Matrix` arithmetic operations

```
function "+" (Right : Real_Matrix) return Real_Matrix;
function "-" (Right : Real_Matrix) return Real_Matrix;
function "abs" (Right : Real_Matrix) return Real_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of `Right`. This is also the standard mathematical operation for matrix identity, negation and absolute value. The index ranges of the result are those of `Right`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

```
function Transpose (X : Real_Matrix) return Real_Matrix;
```

This function returns the transpose of a matrix `X`. The index ranges of the result are `X'Range(2)` and `X'Range(1)` (first and second index respectively).

This function is exact.

```
function "+" (Left, Right : Real_Matrix) return Real_Matrix;
function "-" (Left, Right : Real_Matrix) return Real_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of `Left` and the matching element of `Right`. This is also the standard mathematical operation for matrix addition and subtraction. The index ranges of the result are those of `Left`. The exception `Constraint_Error` is raised if `Left'Length(1)` $\neq$ `Right'Length(1)` or `Left'Length(2)` $\neq$ `Right'Length(2)`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

```
function "*" (Left, Right : Real_Matrix) return Real_Matrix;
```

This operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are `Left'Range(1)` and `Right'Range(2)` (first and second index respectively). The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left, Right : Real_Vector) return Real_Matrix;
```

This operation applies the standard mathematical operation for multiplication of a (column) vector `Left` by a (row) vector `Right`. The index ranges of the matrix result are `Left'Range` and `Right'Range` (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652.

```
function "*" (Left  : Real_Vector;
              Right : Real_Matrix) return Real_Vector;
```

This operation applies the standard mathematical operation for multiplication of a (row) vector `Left` by a matrix `Right`. The index range of the (row) vector result is `Right'Range(2)`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Real_Matrix;
              Right : Real_Vector) return Real_Vector;
```

This operation applies the standard mathematical operation for multiplication of a matrix `Left` by a (column) vector `Right`. The index range of the (column) vector result is `Left'Range(1)`. The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

7.6 `Real_Matrix` scaling operations

```
function "*" (Left  : Real'Base;
              Right : Real_Matrix) return Real_Matrix;
```

This operation applies the standard mathematical operation for scaling a matrix `Right` by a real number `Left`. The index ranges of the matrix result are those of `Right`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of multiplication, as defined in ISO/IEC 8652.

```
function "*" (Left  : Real_Matrix;
              Right : Real'Base) return Real_Matrix;
function "/" (Left  : Real_Matrix;
              Right : Real'Base) return Real_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix `Left` by a real number `Right`. The index ranges of the matrix result are those of `Left`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation, as defined in ISO/IEC 8652.

7.7 Other `Real_Matrix` operations

```
function Identity_Matrix (Order : Positive;
                          First_1, First_2 : Integer := 1) return Real_Matrix;
```

This function returns a square "identity matrix" with $\texttt{Order}^2$ elements and lower bounds of `First_1` and `First_2` (for the first and second index ranges respectively). All elements are set to 0.0 except for the main diagonal, whose elements are set to 1.0. The exception `Constraint_Error` is raised if $\texttt{First\_1} + \texttt{Order} - 1 > \texttt{Integer'Last}$ or $\texttt{First\_2} + \texttt{Order} - 1 > \texttt{Integer'Last}$.

This function is exact.

8 Generic Complex Arrays Package

The generic package `Generic_Complex_Arrays` defines operations and types for complex and mixed real and complex vector and matrix arithmetic. One generic formal type parameter is defined for `Generic_Complex_Arrays`, the floating-point type `Real` which determines the precision of the arithmetic to be used in an instantiation of this generic package. Two formal package parameters, `Real_Arrays` and `Complex_Types` are also defined.

The Ada package specification for `Generic_Complex_Arrays` is given in annex B.

8.1 Types

Two types are defined and exported by `Generic_Complex_Arrays`. The composite type `Complex_Vector` is provided to represent a vector with elements of type `Complex`; it is defined as an unconstrained, one-dimensional array with an index of type `Integer`. The composite type `Complex_Matrix` is provided to represent a matrix with elements of type `Complex`; it is defined as an unconstrained, two-dimensional array with indices of type `Integer`.

8.2 `Complex_Vector` selection, conversion and composition operations

```
function Re (X : Complex_Vector) return Real_Vector;
function Im (X : Complex_Vector) return Real_Vector;
```

Each function returns a vector of the specified cartesian component-parts of `X`. The index range of the result is `X'Range`.

Each function is exact.

```
procedure Set_Re (X  : in out Complex_Vector;
                  Re : in     Real_Vector);
procedure Set_Im (X  : in out Complex_Vector;
                  Im : in     Real_Vector);
```

Each procedure resets the specified (cartesian) component of each of the elements of `X`; the other (cartesian) component of each of the elements is unchanged. The exception `Constraint_Error` is raised if `X'Length` $\neq$ `Re'Length` and if `X'Length` $\neq$ `Im'Length`.

Each procedure is exact.

```
function Compose_From_Cartesian
   (Re : Real_Vector) return Complex_Vector;
function Compose_From_Cartesian
   (Re, Im : Real_Vector) return Complex_Vector;
```

Each function constructs a vector of `Complex` results (in cartesian representation) formed from given vectors of cartesian component-parts (when only the real component-parts are given, imaginary component-parts of zero are assumed). The index range of the result is `Re'Range`. The exception `Constraint_Error` is raised if `Re'Length` $\neq$ `Im'Length`.

Each function is exact.

```
function Modulus (X : Complex_Vector) return Real_Vector;
function "abs" (Right : Complex_Vector) return Real_Vector
   renames Modulus;
function Argument (X : Complex_Vector) return Real_Vector;
function Argument (X     : Complex_Vector;
                   Cycle : Real'Base) return Real_Vector;
```

Each function calculates and returns a vector of the specified polar component-parts of `X`. The index range of the result is `X'Range`. Each array element of the result shall satisfy the (scalar) range definition of the appropriate function.

`Cycle` defines the period of `Argument`; when no `Cycle` is given, a period of 2π is assumed. The exception `Argument_Error` is raised for `Cycle` ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

```
function Compose_From_Polar
   (Modulus, Argument : Real_Vector) return Complex_Vector;
function Compose_From_Polar
   (Modulus, Argument : Real_Vector;
    Cycle             : Real'Base) return Complex_Vector;
```

Each function constructs a vector of `Complex` results (in cartesian representation) formed from given vectors of polar component-parts. Each element of `Argument` is assumed to have a period of `Cycle` (and is reduced accordingly); when no `Cycle` is given, a period of 2π is assumed. The index range of the result is `Modulus'Range`. The exception `Constraint_Error` is raised if `Modulus'Length` $\neq$ `Argument'Length`; the exception `Argument_Error` is raised for `Cycle` ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

8.3 `Complex_Vector` arithmetic operations

```
function "+" (Right : Complex_Vector) return Complex_Vector;
function "-" (Right : Complex_Vector) return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of `Right`. This is also the standard mathematical operation for vector identity and negation. The index range of the result is `Right'Range`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function Conjugate (X : Complex_Vector) return Complex_Vector;
```

This function returns the result of applying the standard mathematical operation for complex conjugation to each element of `X`. The index range of the result is `X'Range`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex conjugation.

```
function "+" (Left, Right : Complex_Vector) return Complex_Vector;
function "-" (Left, Right : Complex_Vector) return Complex_Vector;
function "*" (Left, Right : Complex_Vector) return Complex_Vector;
function "/" (Left, Right : Complex_Vector) return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of `Left` and the matching element of `Right`. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is `Left'Range`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "**" (Left  : Complex_Vector;
               Right : Integer) return Complex_Vector;
```

This operation returns the result of applying the standard mathematical operation for complex exponentiation by an integer power to each element of `Left`. The index range of the result is `Left'Range`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised if for some integer `I` (in the index range of `Left`), `Left(I)` $= (0.0, 0.0)$ and `Right` < 0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex exponentiation by an integer power.

```
function "*" (Left, Right : Complex_Vector) return Complex;
```

This operation returns the inner (dot) product of `Left` and `Right`; no complex conjugation is performed. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

8.4 Mixed `Real_Vector` and `Complex_Vector` arithmetic operations

```
function "+" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "+" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "-" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "-" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "/" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
```

Each operation returns the result of applying the appropriate operation to each element of `Left` and the matching element of `Right`. This is also the standard mathematical operation for vector addition, subtraction, multiplication and division. The index range of the result is `Left'Range`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (real or complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

```
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex;
```

Each operation returns the inner (dot) product of `Left` and `Right`. The exception `Constraint_Error` is raised if `Left'Length` ≠ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

8.5 `Complex_Vector` scaling operations

```
function "*" (Left  : Complex;
              Right : Complex_Vector) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector `Right` by a complex number `Left`. The index range of the result is `Right'Range`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left  : Complex_Vector;
              Right : Complex) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Complex) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a vector `Left` by a complex number `Right`. The index range of the result is `Left'Range`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left  : Real'Base;
              Right : Complex_Vector) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a complex vector `Right` by a real number `Left`. The index range of the result is `Right'Range`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left  : Complex_Vector;
              Right : Real'Base) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real'Base) return Complex_Vector;
```

Each operation applies the standard mathematical operation for scaling a complex vector `Left` by a real number `Right`. The index range of the result is `Left'Range`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (real) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

8.6 Other `Complex_Vector` operations

```
function Unit_Vector (Index : Integer;
                      Order : Positive;
                      First : Integer := 1) return Complex_Vector;
```

This function returns a "unit vector" with `Order` elements and a lower bound of `First`. All elements are set to $(0.0, 0.0)$ except for the `Index` element which is set to $(1.0, 0.0)$. The exception `Constraint_Error` is raised if `Index` $<$ `First`, `Index` $>$ `First` $+$ `Order` $-$ 1, or if `First` $+$ `Order` $-$ 1 $>$ `Integer'Last`.

This function is exact.

8.7 `Complex_Matrix` selection, conversion and composition operations

```
function Re (X : Complex_Matrix) return Real_Matrix;
function Im (X : Complex_Matrix) return Real_Matrix;
```

Each function returns a matrix of the specified cartesian component-parts of `X`. The index ranges of the result are those of `X`.

Each function is exact.

```
procedure Set_Re (X  : in out Complex_Matrix;
                  Re : in     Real_Matrix);
procedure Set_Im (X  : in out Complex_Matrix;
                  Im : in     Real_Matrix);
```

Each procedure resets the specified (cartesian) component of each of the elements of `X`; the other (cartesian) component of each of the elements is unchanged. The exception `Constraint_Error` is raised if `X'Length(1)` $\neq$ `Re'Length(1)` or `X'Length(2)` $\neq$ `Re'Length(2)` and if `X'Length(1)` $\neq$ `Im'Length(1)` or `X'Length(2)` $\neq$ `Im'Length(2)`.

Each procedure is exact.

```
function Compose_From_Cartesian
   (Re : Real_Matrix) return Complex_Matrix;
function Compose_From_Cartesian
   (Re, Im : Real_Matrix) return Complex_Matrix;
```

Each function constructs a matrix of `Complex` results (in cartesian representation) formed from given matrices of cartesian component-parts (when only the real component-parts are given, imaginary component-parts of zero are assumed). The index ranges of the result are those of `Re`. The exception `Constraint_Error` is raised if `Re'Length(1)` $\neq$ `Im'Length(1)` or `Re'Length(2)` $\neq$ `Im'Length(2)`.

Each function is exact.

```
function Modulus (X : Complex_Matrix) return Real_Matrix;
function "abs" (Right : Complex_Matrix) return Real_Matrix
   renames Modulus;
function Argument (X : Complex_Matrix) return Real_Matrix;
function Argument (X     : Complex_Matrix;
                   Cycle : Real'Base) return Real_Matrix;
```

Each function calculates and returns a matrix of the specified polar component-parts of `X`. The index ranges of the result are those of `X`. Each array element of the result shall satisfy the (scalar) range definition of the appropriate function.

`Cycle` defines the period of `Argument`; when no `Cycle` is given, a period of 2π is assumed. The exception `Argument_Error` is raised for `Cycle` ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

```
function Compose_From_Polar
   (Modulus, Argument : Real_Matrix) return Complex_Matrix;
function Compose_From_Polar
   (Modulus, Argument : Real_Matrix;
    Cycle             : Real'Base) return Complex_Matrix;
```

Each function constructs a matrix of `Complex` results (in cartesian representation) formed from given matrices of polar component-parts. Each element of `Argument` is assumed to have a period of `Cycle` (and is reduced accordingly); when no `Cycle` is given, a period of 2π is assumed. The index ranges of the result are those of `Modulus`. The exception `Constraint_Error` is raised if `Modulus'Length(1)` $\neq$ `Argument'Length(1)` or `Modulus'Length(2)` $\neq$ `Argument'Length(2)`; the exception `Argument_Error` is raised for `Cycle` ≤ 0.0.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate function.

8.8 `Complex_Matrix` arithmetic operations

```
function "+" (Right : Complex_Matrix) return Complex_Matrix;
function "-" (Right : Complex_Matrix) return Complex_Matrix;
```

Each operation returns the result of applying the appropriate operation to each element of `Right`. This is also the standard mathematical operation for matrix identity and negation. The index ranges of the result are those of `Right`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function Conjugate (X : Complex_Matrix) return Complex_Matrix;
```

This function returns the result of applying the standard mathematical operation for complex conjugation to each element of `X`. The index ranges of the result are those of `X`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex conjugation.

```
function Transpose (X : Complex_Matrix) return Complex_Matrix;
```

This function returns the transpose of a matrix `X`. The index ranges of the result are `X'Range(2)` and `X'Range(1)` (first and second index respectively).

This function is exact.

```
function "+" (Left, Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left, Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the appropriate standard mathematical operation for matrix addition or subtraction. The index ranges of the result are those of `Left`. The exception `Constraint_Error` is raised if `Left'Length(1)` $\neq$ `Right'Length(1)` or `Left'Length(2)` $\neq$ `Right'Length(2)`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left, Right : Complex_Matrix) return Complex_Matrix;
```

This operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are `Left'Range(1)` and `Right'Range(2)` (first and second index respectively). The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left, Right : Complex_Vector) return Complex_Matrix;
```

This operation applies the standard mathematical operation for multiplication of a (column) vector by a (row) vector. The index ranges of the matrix result are `Left'Range` and `Right'Range` (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left  : Complex_Vector;
              Right : Complex_Matrix) return Complex_Vector;
```

This operation applies the standard mathematical operation for multiplication of a (row) vector by a matrix. The index range of the (row) vector result is `Right'Range(2)`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Complex_Matrix;
              Right : Complex_Vector) return Complex_Vector;
```

This operation applies the standard mathematical operation for multiplication of a matrix by a (column) vector. The index range of the (column) vector result is `Left'Range(1)`. The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

8.9 Mixed `Real_Matrix` and `Complex_Matrix` arithmetic operations

```
function "+" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "+" (Left  : Complex_Matrix;
              Right : Real_Matrix) return Complex_Matrix;
function "-" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "-" (Left  : Complex_Matrix;
              Right : Real_Matrix) return Complex_Matrix;
```

Each operation applies the appropriate standard mathematical operation for matrix addition or subtraction. The index ranges of the result are those of `Left`. The exception `Constraint_Error` is raised if `Left'Length(1)` $\neq$ `Right'Length(1)` or `Left'Length(2)` $\neq$ `Right'Length(2)`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

```
function "*" (Left  : Real_Matrix;
              Right : Complex_Matrix) return Complex_Matrix;
function "*" (Left  : Complex_Matrix;
              Right : Real_Matrix) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for matrix multiplication. The index ranges of the result are `Left'Range(1)` and `Right'Range(2)` (first and second index respectively). The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Matrix;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for multiplication of a (column) vector by a (row) vector. The index ranges of the matrix result are `Left'Range` and `Right'Range` (first and second index respectively).

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left  : Real_Vector;
              Right : Complex_Matrix) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Matrix) return Complex_Vector;
```

Each operation applies the standard mathematical operation for multiplication of a (row) vector by a matrix. The index range of the (row) vector result is `Right'Range(2)`. The exception `Constraint_Error` is raised if `Left'Length` $\neq$ `Right'Length(1)`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

```
function "*" (Left  : Real_Matrix;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Matrix;
              Right : Real_Vector) return Complex_Vector;
```

Each operation applies the standard mathematical operation for multiplication of a matrix by a (column) vector. The index range of the (column) vector result is `Left'Range(1)`. The exception `Constraint_Error` is raised if `Left'Length(2)` $\neq$ `Right'Length`.

This operation involves an inner product; an accuracy requirement is not specified.

Clause 6 applies when the elements of `Left` and `Right` are such that computation of an intermediate result could signal overflow.

8.10 `Complex_Matrix` scaling operations

```
function "*" (Left  : Complex;
              Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix `Right` by a complex number `Left`. The index ranges of the result are those of `Right`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of complex multiplication.

```
function "*" (Left  : Complex_Matrix;
              Right : Complex) return Complex_Matrix;
function "/" (Left  : Complex_Matrix;
              Right : Complex) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a matrix `Left` by a complex number `Right`. The index ranges of the result are those of `Left`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (complex) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for complex arithmetic.

```
function "*" (Left  : Real'Base;
              Right : Complex_Matrix) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a complex matrix `Right` by a real number `Left`. The index ranges of the result are those of `Right`.

Each array element of the result shall satisfy the (scalar) accuracy requirement of mixed real and complex multiplication.

```
function "*" (Left  : Complex_Matrix;
              Right : Real'Base) return Complex_Matrix;
function "/" (Left  : Complex_Matrix;
              Right : Real'Base) return Complex_Matrix;
```

Each operation applies the standard mathematical operation for scaling a complex matrix `Left` by a real number `Right`. The index ranges of the result are those of `Left`. The exception specified by ISO/IEC 8652 for signaling division by zero is raised when division by (real) zero is attempted.

Each array element of the result shall satisfy the (scalar) accuracy requirement of the appropriate operation for mixed real and complex arithmetic.

8.11 Other `Complex_Matrix` operations

```
function Identity_Matrix (Order : Positive;
                          First_1, First_2 : Integer := 1) return Complex_Matrix;
```

This function returns a square "identity matrix" with $\texttt{Order}^2$ elements and lower bounds of `First_1` and `First_2` (for the first and second index ranges respectively). All elements are set to $(0.0, 0.0)$ except for the main diagonal, whose elements are set to $(1.0, 0.0)$. The exception `Constraint_Error` is raised if $\texttt{First\_1} + \texttt{Order} - 1 > \texttt{Integer'Last}$ or $\texttt{First\_2} + \texttt{Order} - 1 > \texttt{Integer'Last}$.

This function is exact.

9 Standard non-generic packages

In addition to the generic type packages, analogous non-generic packages are required to define standard real and complex vector and matrix types. Non-generic packages shall be provided for all precisions defined in package `Standard`. The same floating-point type shall be used to generate real and complex packages of the same precision.

The packages `Real_Arrays` and `Complex_Arrays` shall always be provided; these packages shall define the same types and subprograms as `Generic_Real_Arrays` and `Generic_Complex_Arrays`, respectively, except that the predefined type `Float` shall replace type `Real` throughout.

Names of the other non-generic packages (where defined) shall be assigned as follows:

— if the predefined floating-point type `Short_Float` is supported by a host implementation of ISO/IEC 8652, then this type shall be used to generate the packages `Short_Real_Arrays` and `Short_Complex_Arrays`;

— if the predefined floating-point type `Long_Float` is supported by a host implementation of ISO/IEC 8652, then this type shall be used to generate the packages `Long_Real_Arrays` and `Long_Complex_Arrays`; and

— if other predefined floating-point types are supported (e.g., `Long_Long_Float`), package names shall be assigned by considering the predefined types in order of ascending (for `Long`-types) or descending (for `Short`-types) precision and matching the prefix of each floating-point type with that of the corresponding package names.

Each non-generic package shall define the same types and subprograms as the corresponding generic package, except that the appropriate predefined type shall replace type `Real` throughout.

The nongeneric equivalent packages may, but need not, be actual instantiations of the generic package for the appropriate predefined type.

Annex A
(normative)
Ada specification for `Generic_Real_Arrays`

```
generic
   type Real is digits <>;
package Ada.Numerics.Generic_Real_Arrays is
   pragma Pure(Generic_Real_Arrays);

-- Types

   type Real_Vector is array (Integer range <>) of Real'Base;
   type Real_Matrix is array (Integer range <>,
                              Integer range <>) of Real'Base;

-- Subprograms for Real_Vector Types

   -- Real_Vector arithmetic operations

   function "+" (Right : Real_Vector) return Real_Vector;
   function "-" (Right : Real_Vector) return Real_Vector;
   function "abs" (Right : Real_Vector) return Real_Vector;

   function "+" (Left, Right : Real_Vector) return Real_Vector;
   function "-" (Left, Right : Real_Vector) return Real_Vector;
   function "*" (Left, Right : Real_Vector) return Real_Vector;
   function "/" (Left, Right : Real_Vector) return Real_Vector;
   function "**" (Left  : Real_Vector;
                  Right : Integer) return Real_Vector;

   function "*" (Left, Right : Real_Vector) return Real'Base;

   -- Real_Vector scaling operations

   function "*" (Left  : Real'Base;
                 Right : Real_Vector) return Real_Vector;
   function "*" (Left  : Real_Vector;
                 Right : Real'Base) return Real_Vector;
   function "/" (Left  : Real_Vector;
                 Right : Real'Base) return Real_Vector;

   -- Other Real_Vector operations

   function Unit_Vector (Index : Integer;
                         Order : Positive;
                         First : Integer := 1) return Real_Vector;

-- Subprograms for Real_Matrix Types

   -- Real_Matrix arithmetic operations
```

```
   function "+" (Right : Real_Matrix) return Real_Matrix;
   function "-" (Right : Real_Matrix) return Real_Matrix;
   function "abs" (Right : Real_Matrix) return Real_Matrix;
   function Transpose (X : Real_Matrix) return Real_Matrix;

   function "+" (Left, Right : Real_Matrix) return Real_Matrix;
   function "-" (Left, Right : Real_Matrix) return Real_Matrix;
   function "*" (Left, Right : Real_Matrix) return Real_Matrix;

   function "*" (Left, Right : Real_Vector) return Real_Matrix;
   function "*" (Left  : Real_Vector;
                 Right : Real_Matrix) return Real_Vector;
   function "*" (Left  : Real_Matrix;
                 Right : Real_Vector) return Real_Vector;

   -- Real_Matrix scaling operations

   function "*" (Left  : Real'Base;
                 Right : Real_Matrix) return Real_Matrix;
   function "*" (Left  : Real_Matrix;
                 Right : Real'Base) return Real_Matrix;
   function "/" (Left  : Real_Matrix;
                 Right : Real'Base) return Real_Matrix;

   -- Other Real_Matrix operations

   function Identity_Matrix (Order : Positive;
                             First_1, First_2 : Integer := 1) return Real_Matrix;

end Ada.Numerics.Generic_Real_Arrays;
```

Annex B
(normative)
Ada specification for Generic_Complex_Arrays

```
generic
   with package Real_Arrays is new Ada.Numerics.Generic_Real_Arrays (<>);
   with package Complex_Types is new Ada.Numerics.Generic_Complex_Types (<>);
   use Real_Arrays;
   use Complex_Types;
package Ada.Numerics.Generic_Complex_Arrays is
   pragma Pure(Generic_Complex_Arrays);

-- Types

   type Complex_Vector is array (Integer range <>) of Complex;
   type Complex_Matrix is array (Integer range <>,
                                 Integer range <>) of Complex;

-- Subprograms for Complex_Vector types

   -- Complex_Vector selection, conversion and composition operations

   function Re (X : Complex_Vector) return Real_Vector;
   function Im (X : Complex_Vector) return Real_Vector;

   procedure Set_Re (X  : in out Complex_Vector;
                     Re : in     Real_Vector);
   procedure Set_Im (X  : in out Complex_Vector;
                     Im : in     Real_Vector);

   function Compose_From_Cartesian
      (Re : Real_Vector) return Complex_Vector;
   function Compose_From_Cartesian
      (Re, Im : Real_Vector) return Complex_Vector;

   function Modulus (X : Complex_Vector) return Real_Vector;
   function "abs" (Right : Complex_Vector) return Real_Vector
      renames Modulus;
   function Argument (X : Complex_Vector) return Real_Vector;
   function Argument (X     : Complex_Vector;
                      Cycle : Real'Base) return Real_Vector;

   function Compose_From_Polar
      (Modulus, Argument : Real_Vector) return Complex_Vector;
   function Compose_From_Polar
      (Modulus, Argument : Real_Vector;
       Cycle             : Real'Base) return Complex_Vector;

   -- Complex_Vector arithmetic operations
```

```
function "+" (Right : Complex_Vector) return Complex_Vector;
function "-" (Right : Complex_Vector) return Complex_Vector;
function Conjugate (X : Complex_Vector) return Complex_Vector;

function "+" (Left, Right : Complex_Vector) return Complex_Vector;
function "-" (Left, Right : Complex_Vector) return Complex_Vector;
function "*" (Left, Right : Complex_Vector) return Complex_Vector;
function "/" (Left, Right : Complex_Vector) return Complex_Vector;
function "**" (Left  : Complex_Vector;
               Right : Integer) return Complex_Vector;

function "*" (Left, Right : Complex_Vector) return Complex;

-- Mixed Real_Vector and Complex_Vector arithmetic operations

function "+" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "+" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "-" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "-" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;
function "/" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex_Vector;

function "*" (Left  : Real_Vector;
              Right : Complex_Vector) return Complex;
function "*" (Left  : Complex_Vector;
              Right : Real_Vector) return Complex;

-- Complex_Vector scaling operations

function "*" (Left  : Complex;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Complex) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Complex) return Complex_Vector;

function "*" (Left  : Real'Base;
              Right : Complex_Vector) return Complex_Vector;
function "*" (Left  : Complex_Vector;
              Right : Real'Base) return Complex_Vector;
function "/" (Left  : Complex_Vector;
              Right : Real'Base) return Complex_Vector;
```

```
   -- Other Complex_Vector operations

   function Unit_Vector (Index : Integer;
                         Order : Positive;
                         First : Integer := 1) return Complex_Vector;

-- Subprograms for Complex_Matrix Types

   -- Complex_Matrix selection, conversion and composition operations

   function Re (X : Complex_Matrix) return Real_Matrix;
   function Im (X : Complex_Matrix) return Real_Matrix;

   procedure Set_Re (X  : in out Complex_Matrix;
                     Re : in     Real_Matrix);
   procedure Set_Im (X  : in out Complex_Matrix;
                     Im : in     Real_Matrix);

   function Compose_From_Cartesian
      (Re : Real_Matrix) return Complex_Matrix;
   function Compose_From_Cartesian
      (Re, Im : Real_Matrix) return Complex_Matrix;

   function Modulus (X : Complex_Matrix) return Real_Matrix;
   function "abs" (Right : Complex_Matrix) return Real_Matrix
      renames Modulus;

   function Argument (X : Complex_Matrix) return Real_Matrix;
   function Argument (X     : Complex_Matrix;
                      Cycle : Real'Base) return Real_Matrix;

   function Compose_From_Polar
      (Modulus, Argument : Real_Matrix) return Complex_Matrix;
   function Compose_From_Polar
      (Modulus, Argument : Real_Matrix;
       Cycle             : Real'Base) return Complex_Matrix;

   -- Complex_Matrix arithmetic operations

   function "+" (Right : Complex_Matrix) return Complex_Matrix;
   function "-" (Right : Complex_Matrix) return Complex_Matrix;
   function Conjugate (X : Complex_Matrix) return Complex_Matrix;
   function Transpose (X : Complex_Matrix) return Complex_Matrix;

   function "+" (Left, Right : Complex_Matrix) return Complex_Matrix;
   function "-" (Left, Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left, Right : Complex_Matrix) return Complex_Matrix;

   function "*" (Left, Right : Complex_Vector) return Complex_Matrix;
   function "*" (Left  : Complex_Vector;
                 Right : Complex_Matrix) return Complex_Vector;
```

```
   function "*" (Left  : Complex_Matrix;
                 Right : Complex_Vector) return Complex_Vector;

   -- Mixed Real_Matrix and Complex_Matrix arithmetic operations

   function "+" (Left  : Real_Matrix;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "+" (Left  : Complex_Matrix;
                 Right : Real_Matrix) return Complex_Matrix;
   function "-" (Left  : Real_Matrix;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "-" (Left  : Complex_Matrix;
                 Right : Real_Matrix) return Complex_Matrix;
   function "*" (Left  : Real_Matrix;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left  : Complex_Matrix;
                 Right : Real_Matrix) return Complex_Matrix;

   function "*" (Left  : Real_Vector;
                 Right : Complex_Vector) return Complex_Matrix;
   function "*" (Left  : Complex_Vector;
                 Right : Real_Vector) return Complex_Matrix;
   function "*" (Left  : Real_Vector;
                 Right : Complex_Matrix) return Complex_Vector;
   function "*" (Left  : Complex_Vector;
                 Right : Real_Matrix) return Complex_Vector;
   function "*" (Left  : Real_Matrix;
                 Right : Complex_Vector) return Complex_Vector;
   function "*" (Left  : Complex_Matrix;
                 Right : Real_Vector) return Complex_Vector;

   -- Complex_Matrix scaling operations

   function "*" (Left  : Complex;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left  : Complex_Matrix;
                 Right : Complex) return Complex_Matrix;
   function "/" (Left  : Complex_Matrix;
                 Right : Complex) return Complex_Matrix;

   function "*" (Left  : Real'Base;
                 Right : Complex_Matrix) return Complex_Matrix;
   function "*" (Left  : Complex_Matrix;
                 Right : Real'Base) return Complex_Matrix;
   function "/" (Left  : Complex_Matrix;
                 Right : Real'Base) return Complex_Matrix;

   -- Other Complex_Matrix operations

   function Identity_Matrix (Order : Positive;
                             First_1, First_2 : Integer := 1) return Complex_Matrix;
```

```
end Ada.Numerics.Generic_Complex_Arrays;
```

Annex C
(informative)
Rationale

Annex D
(informative)
Bibliography