

Information technology — Programming languages — Generic package of complex elementary functions for Ada

ICS 35.060

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW



Document Sharing Hub

National foreword

This British Standard reproduces verbatim ISO/IEC 13814:1998 and implements it as the UK national standard.

The UK participation in its preparation was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interface, which has the responsibility to:

- aid enquirers to understand the text;
- present to the responsible international/European committee any enquiries on the interpretation, or proposals for change, and keep the UK interests informed;
- monitor related international and European developments and promulgate them in the UK.

A list of organizations represented on this committee can be obtained on request to its secretary.

Cross-references

The British Standards which implement international or European publications referred to in this document may be found in the BSI Standards Catalogue under the section entitled “International Standards Correspondence Index”, or by using the “Find” facility of the BSI Standards Electronic Catalogue.

A British Standard does not purport to include all the necessary provisions of a contract. Users of British Standards are responsible for their correct application.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Summary of pages

This document comprises a front cover, an inside front cover, pages i and ii, the ISO/IEC title page, pages ii to iv, pages 1 to 28 and a back cover.

This standard has been updated (see copyright date) and may have had amendments incorporated. This will be indicated in the amendment table on the inside front cover.

This British Standard, having been prepared under the direction of the DISC Board, was published under the authority of the Standards Board and comes into effect on 15 September 1998

© BSI 04-1999

ISBN 0 580 28865 X

Amendments issued since publication

Amd. No.	Date	Comments

Contents

	Page
National foreword	Inside front cover
Foreword	iv
Text of ISO/IEC 13814	1

INTERNATIONAL
STANDARD

ISO/IEC
13814

First edition
1998-07-01

**Information technology — Programming
languages — Generic package of complex
elementary functions for Ada**

*Technologies de l'information — Langages de programmation —
Paquetage générique de fonctions élémentaires complexes pour Ada*



Contents

	Page
Foreword	iv
Introduction	1
1 Scope	1
2 Normative references	1
3 Subprograms provided	1
4 Instantiations	2
5 Implementations	3
6 Exceptions	4
7 Arguments outside the range of safe numbers	5
8 Method of specification of functions	3
9 Branch cut and domain definitions	5
10 Range definitions	5
11 Accuracy requirements	6
12 Overflow	7
13 Underflow	8
14 Specifications of the functions	8
14.1 SQRT — Square root	8
14.2 LOG — Natural logarithm	9
14.3 EXP — Exponential function, complex argument	9
14.4 EXP — Exponential function, imaginary argument	10
14.5 “**” — Exponentiation operator, complex to complex power arguments	10
14.6 “**” — Exponentiation operator, complex to real power arguments	10
14.7 “**” — Exponentiation operator, real to complex power arguments	11
14.8 SIN — Trigonometric sine function	11
14.9 COS — Trigonometric cosine function	12
14.10 TAN — Trigonometric tangent function	12
14.11 COT — Trigonometric cotangent function	13
14.12 ARCSIN — Inverse trigonometric sine function	13
14.13 ARCCOS — Inverse trigonometric cosine function	14
14.14 ARCTAN — Inverse trigonometric tangent function	14
14.15 ARCCOT — Inverse trigonometric cotangent function	15
14.16 SINH — Hyperbolic sine function	15
14.17 COSH — Hyperbolic cosine function	16
14.18 TANH — Hyperbolic tangent function	16
14.19 COTH — Hyperbolic cotangent function	17
14.20 ARCSINH — Inverse hyperbolic sine function	17
14.21 ARCCOSH — Inverse hyperbolic cosine function	17
14.22 ARCTANH — Inverse hyperbolic tangent function	18
14.23 ARCCOTH — Inverse hyperbolic cotangent function	18
Annex A (normative) Ada specification for GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS	20
Annex B (informative) Rationale	21
B.1 Introduction and motivation	21
B.2 History	21
B.3 Packaging considerations	22
B.4 Choice of functions	23
B.5 Handling of exceptions	23
B.6 Matters of style	24
B.7 Method of expressing accuracy requirements	25

	Page
Annex C (informative) Bibliography	27
Figure B.1 — Result rectangle for a subprogram satisfying a maximum relative component error requirement	26
Figure B.2 — Result rectangle for a subprogram satisfying a maximum relative box error requirement	26
Figure B.3 — Result circle for a subprogram satisfying a maximum relative vector error requirement	27

Descriptors: Data processing, computer software, artificial languages, programming languages, advanced language, Ada, implementation.

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 13814 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee 22, *Programming languages, their environments and system software interfaces*.

Annex A forms an integral part of this International Standard. Annex B and Annex C are for information only.

Introduction

The generic package described here is intended to provide the basic complex mathematical routines 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 specification included in this International Standard is presented as a compilable Ada specification in Annex A, with explanatory text in numbered clauses in the main body of text. The explanatory text is normative, with the exception of the following items:

- notes (under the heading Notes associated with some of the functions); and
- notes (labeled as such) presented at the end of any numbered clause.

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.

1 Scope

This International Standard defines the specification of a generic package of complex elementary functions called `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`. It does not provide the body of the package.

This International Standard specifies certain basic complex mathematical routines 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.

This International Standard is applicable to programming environments conforming to ISO/IEC 8652:1987 and is relevant to the revised standard ISO/IEC 8652:1995.

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*.

ISO/IEC 11430, *Information technology — Programming languages — Generic package of elementary functions for Ada*.

ISO/IEC 11729, *Information technology — Programming languages Generic package of primitive functions for Ada*.

ISO/IEC 13813, *Information technology — Programming languages — Generic packages of real and complex type declarations and basic operations for Ada (including vector and matrix types)*.

3 Subprograms provided

The following twenty mathematical functions are provided:

SQRT	LOG	EXP	“**”
SIN	COS	TAN	COT
ARCSIN	ARCCOS	ARCTAN	ARCCOT
SINH	COSH	TANH	COTH
ARCSINH	ARCCOSH	ARCTANH	ARCCOTH

These are the square root (SQRT), logarithm (LOG) and exponential (EXP) function and the exponentiation operator (**); the trigonometric functions for sine (SIN), cosine (COS), tangent (TAN) and cotangent (COT) and their inverses (ARCSIN, ARCCOS, ARCTAN and ARCCOT); and the hyperbolic functions for sine (SINH), cosine (COSH), tangent (TANH), and cotangent (COTH) together with their inverses (ARCSINH, ARCCOSH, ARCTANH, and ARCCOTH). These are the same functions by the same names that are in the package `GENERIC_ELEMENTARY_FUNCTIONS` defined in ISO/IEC 11430.

Several variations are provided for the exponentiation operator and the exponential function. All functions have one or more formal parameters of type `COMPLEX` or `IMAGINARY` and return a value of type `COMPLEX`. `COMPLEX` and `IMAGINARY` are generic formal parameters of the generic package `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`.

4 Instantiations

This International Standard describes a generic package, `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`, which the user must instantiate to obtain a computational capability. The generic package has three required generic formal parameters and many generic formal subprograms with defaults. At instantiation, the user must specify, as the generic actual parameters to be associated with `REAL`, `COMPLEX` and `IMAGINARY` respectively,

- a) a floating-point type,
- b) a private composite type containing real and imaginary components of the type specified by a), (Within this International Standard, the real and imaginary components of the composite type are referred to simply as components.)
- c) a private type of the same base type as a) that is interpreted as the pure imaginary form of b).

Types suitable for b) and c) are exported by the package obtained by instantiating, with the type specified by a) as its generic actual parameter, the generic package `GENERIC_COMPLEX_TYPES` defined in ISO/IEC 13813. (Instantiation of the package `GENERIC_COMPLEX_TYPES` is not a prerequisite for using `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`; it is merely an option that the user might find convenient to pursue.) In addition the user may specify the many subprogram generic formal parameters as generic actual parameters, or for convenience, all these subprograms are exported by the instantiation of `GENERIC_COMPLEX_TYPES` described above.

Depending on the implementation, the user may or may not be allowed to specify a generic actual type having a range constraint for the first generic actual parameter (see clause 5). The generic actual parameters for the subprograms can be omitted if functions having names and profiles matching those of the corresponding generic formal parameters are visible at the place of the instantiation. An instantiation of `GENERIC_COMPLEX_TYPES`, as described above, can optionally be used to obtain subprograms that satisfy these requirements.

In addition to the body of the generic package itself, implementers may provide (non-generic) library packages that can be used just like instantiations of the generic package in which the first generic actual parameter is a predefined floating-point type and the remainder are related to the first in appropriate ways. In particular, the name of a package serving as a replacement for an instantiation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` for the predefined type `FLOAT` shall be `COMPLEX_ELEMENTARY_FUNCTIONS` with `REAL` replaced by `FLOAT` in the profiles of the functions that it exports. Similarly, the names of packages serving as replacements for instantiations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` for the predefined types `LONG_FLOAT`, `SHORT_FLOAT`, etc., shall be `LONG_COMPLEX_ELEMENTARY_FUNCTIONS`, `SHORT_COMPLEX_ELEMENTARY_FUNCTIONS`, etc., respectively and `REAL` shall be systematically replaced in the profiles of the functions that they export by `LONG_FLOAT`, `SHORT_FLOAT`, etc. Implementers are responsible for ensuring that a composite type called `COMPLEX` containing a pair of components of types `FLOAT`, `LONG_FLOAT`, `SHORT_FLOAT`, etc., respectively, are available. For example, each of the non-generic library packages could be prefixed by an appropriate context clause, such as:

```
with COMPLEX_TYPES;  
package COMPLEX_ELEMENTARY_FUNCTIONS is . . .
```

The packages `COMPLEX_TYPES`, `LONG_COMPLEX_TYPES`, etc., described in ISO/IEC 13813, are not a prerequisite for implementation, yet are an option the implementer might find convenient to pursue.

5 Implementations

Portable implementations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` are strongly encouraged. However, implementations are not required to be portable. In particular, an implementation of this International Standard in Ada may use pragma `INTERFACE` or other pragmas, unchecked conversion, machine-code insertions or other machine-dependent techniques as desired. On the other hand, to the extent that generic packages (e.g. `GENERIC_ELEMENTARY_FUNCTIONS` defined in ISO/IEC 11430 and `GENERIC_PRIMITIVE_FUNCTIONS` defined in ISO/IEC 11729) become widely available, portable implementations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` should not be particularly difficult to construct. In particular, it is possible to achieve the accuracy required by this International Standard, when specified, by appropriate use of the package `GENERIC_ELEMENTARY_FUNCTIONS`.

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. An implementation is also allowed to make other reasonable assumptions about the environment in which it is to be used, but only when necessary in order to match algorithms to hardware characteristics in an economical manner. All such limits and assumptions shall be clearly documented. By convention, an implementation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` 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.

An implementation is allowed to impose a restriction that the generic actual parameter associated with `REAL` shall not have a range constraint that reduces the range of allowable values. If it does impose this restriction, then the restriction shall be documented, and the effects of violating the restriction shall be one of the following:

- a) Compilation of a unit containing an instantiation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` is rejected.
- b) `CONSTRAINT_ERROR` or `PROGRAM_ERROR` is raised during the elaboration of an instantiation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`.

Conversely, if an implementation does not impose the restriction, then it must not allow such a range constraint, when included with the user's actual type, to interfere with the internal computations of the functions; that is, if the components of the argument and the result are within the range of the type, then the implementation shall return the result and shall not raise an exception (such as `CONSTRAINT_ERROR`).

Any of the restrictions discussed above may in fact be inherited from implementations of the packages `GENERIC_ELEMENTARY_FUNCTIONS` defined in ISO/IEC 11430, `GENERIC_PRIMITIVE_FUNCTIONS` defined in ISO/IEC 11729 and `GENERIC_COMPLEX_TYPES` defined in ISO/IEC 13813, if used. The dependence of an implementation on such inherited restrictions should be documented.

An implementation shall function properly in a tasking environment. Apart from the obvious restriction that an implementation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` shall avoid declaring variables that are global to the functions, no special constraints are imposed on implementations. Nothing in this International Standard requires the use of such global variables.

Some hardware and their accompanying Ada implementations have the capability of representing and discriminating between positively and negatively signed zeros as a means, e.g., of preserving the sign of an infinitesimal quantity that has underflowed to zero. This International Standard allows implementations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` to exploit that capability, when available, so as to exhibit continuity in the results of some functions as certain limits are approached. At the same time, it accommodates implementations in which that capability is unavailable. Because a definition of what comprises the capability of representing and distinguishing signed zeros is beyond the scope of this International Standard, implementations are allowed the freedom not to exploit the capability, even when available. An implementation shall exercise its choice consistently, either exploiting signed-zero behavior everywhere or nowhere in this package. The signs of zero results prescribed by this International Standard apply only to implementations that exploit signed zeros; an implementation that does not exploit signed zeros may give any sign to a zero result. In addition an implementation shall document its behavior with respect to signed zeros. In this International Standard, unless otherwise qualified, zero means any of $(\pm 0.0, \pm 0.0)$ and one means either of $(1.0, \pm 0.0)$.

6 Exceptions

The `ARGUMENT_ERROR` exception is raised by a function in the generic package when an argument of the function violates one or more of the conditions given in the function's domain definition (see clause 9). Note that these conditions are related only to the mathematical definition of the function and are therefore implementation independent.

The `ARGUMENT_ERROR` exception is declared as a renaming of the exception of the same name declared in the package `ELEMENTARY_FUNCTIONS_EXCEPTIONS` defined in ISO/IEC 11430. This exception distinguishes neither between different kinds of argument errors, nor between different functions, nor between different instantiations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`, nor between instantiations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` and those of `GENERIC_ELEMENTARY_FUNCTIONS`.

Besides `ARGUMENT_ERROR`, the only exceptions allowed during a call to a function in `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` are predefined exceptions, as follows:

- a) Virtually any predefined exception is possible during the evaluation of an argument of a function in `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`. For example, `NUMERIC_ERROR`, `CONSTRAINT_ERROR` or even `PROGRAM_ERROR` could be raised if an argument has an undefined value; and `CONSTRAINT_ERROR` will be raised when the value of an argument (or component of an argument) lies outside the range of the user's generic actual type `REAL`. Additionally `STORAGE_ERROR` could be raised, e.g. if insufficient storage is available to perform the call. All these exceptions are raised before the body of the function is entered and therefore have no bearing on implementations of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`.
- b) Also, `CONSTRAINT_ERROR` will be raised when a function in `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` attempts to construct a complex return value with a component outside the range of the user's generic actual type associated with `REAL`. The exception raised for this reason shall be propagated to the caller of the function.
- c) Whenever the arguments of a function are such that a component of a result permitted by the accuracy requirements would exceed `REAL 'SAFE_LARGE` in absolute value, as formalized below in clause 12, an implementation may raise, and shall then propagate to the caller, the exception specified by Ada for signaling overflow.
- d) Whenever the arguments of a function are such that the corresponding mathematical function is infinite (i.e., has an infinite component), an implementation shall raise and propagate to the caller the exception specified by Ada for signaling division by zero.
- e) Once execution of a function has begun, an implementation may propagate `STORAGE_ERROR` to the caller of the function, but only to signal the unexpected exhaustion of storage. Similarly, once execution of a function has begun, an implementation may propagate `PROGRAM_ERROR` to the caller of the function, but only to signal errors made by the user of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`.

No exception is allowed during a call to a function in `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS` except those permitted by the foregoing rules. In particular, for arguments for which all components of results satisfying the accuracy requirements remain less than or equal to `REAL 'SAFE_LARGE` in absolute value in both real and imaginary parts, a function shall locally handle an overflow occurring during the computation of an intermediate result, if such an overflow is possible, and shall not propagate an exception signaling that overflow to the caller of the function.

The only exceptions allowed during an instantiation of `GENERIC_COMPLEX_ELEMENTARY_FUNCTIONS`, including the execution of the optional sequence of statements in the body of the instance, are `CONSTRAINT_ERROR`, `STORAGE_ERROR` and `PROGRAM_ERROR`, and then only for the following reasons. The raising of `CONSTRAINT_ERROR` during instantiation is only allowed when the implementation imposes the restriction that the generic actual type associated with `REAL` shall not have a range constraint, and the user violates that restriction (it may, in fact, be an inescapable consequence of the violation). The raising of `PROGRAM_ERROR` during instantiation is only allowed for the purpose of signaling errors made by the user violating some restriction or limitation of the implementation. The raising of `STORAGE_ERROR` during instantiation is only allowed for the purpose of signaling the exhaustion of storage.

NOTE In ISO/IEC 8652:1995, the exception specified for signaling overflow is `CONSTRAINT_ERROR`.