Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Kok

Design and implementation of elementary functions in Ada

Department of Numerical Mathematics Report NM-R8710 April

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Design and Implementation of Elementary Functions in Ada

Jan Kok
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

This report describes the design and implementation of the elementary functions in Ada for the EC-funded project "Pilot Implementations of Basic Modules for Large Portable Numerical Libraries in Ada". It presents the specification of a generic package for the declaration of the elementary functions in Ada which employs the project's library error handling mechanism. Further, it describes the portable implementation of this package, and the work done on testing and documenting the package.

1980 Mathematics Subject Classifications: 69D49, 65-04.
Key Words & Phrases: Ada, high level language, basic mathematical functions, scientific libraries, portability.
Note: This report will be submitted for publication elsewhere.
The work described in this report was part of the MAP 750 project part-funded by the Commission of the European Communities. This project, the PIA project, was carried out during 1985-1987 by a consortium consisting of: NAG Ltd, Oxford (Prime Contractor), CWI, Amsterdam, NPL, Teddington, and NIHE, Dublin for Trinity College Dublin, with University of Liverpool as a subcontractor.

CONTENTS

Report NM-R8710
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

DESIGN AND IMPLEMENTATION OF ELEMENTARY FUNCTIONS IN ADA

1. INTRODUCTION

The main objective of the project "Pilot Implementations of Basic Modules for Large Portable Numerical Libraries in Ada" (for short: PIA project) was to design and implement the foundations of a portable and efficient large scale numerical library in Ada for application by both the scientific and real-time computing communities.

In particular, having regard to the published Guidelines for the Design of Large Modular Scientific Libraries in Ada (Symm et al., 1984, also published as Chapter 10 in: Ford et al., 1986) ('Guidelines' for short), basic functions core modules have been designed and implemented, satisfying requirements of general usefulness, ease of use, efficiency in use, and transportability, without too much obstructing their efficient execution.

In this report the PIA project's core module design for the basic mathematical functions is described, and also experience with the implementation of these basic mathematical functions. The result is an Ada package called NAG_GENERIC_S01BA. The specification of this package is derived from the current version of the elementary functions specification as it is now pursued by members of the Ada-Europe Numerics Working Group (see Kok, 1987). It differs from the proposed specification in this respect that the specification (and implementation) of NAG_GENERIC_S01BA employs the Ada library error mechanism which is another result of the PIA project (see NAG_Ada, 1987).

2. STATE OF THE ART

One of the recommendations of the Guidelines was that a standard specification of the basic mathematical functions should be adopted as soon as possible. In these Guidelines, and also in a separate paper (Kok & Symm, 1984), such a standard specification was proposed with an outline of the several possibilities considered and with justification for the options chosen.

For complete details we refer to the above mentioned Guidelines report and paper.

With the present proliferation of new Ada compilers, the need for elementary mathematical functions is now well-recognised. However, inquiries have already shown that nearly every manufacturer (with the exception of those that do not provide elementary functions at all) provides his own set of elementary functions, usually only for the available pre-defined floating-point types of the system, and all provisions differ completely in :-

- contents of the package(s) of basic functions,

- naming of the available functions,

- choice of parameters and their possible defaults,

- names of parameters,

- types of parameters and of the function results, and the use of subtypes,

- numbers, constants, types and subtypes provided along with the functions,

- exceptions which can be raised by the functions.

The above summary reports only on the differences concerning the specification of the elementary functions provided. It is hardly necessary to add that implementations (the function bodies) also differ with respect to :-

- range of applicability,

- behaviour for out-of-range input parameters,

- accuracy obtained, as compared with the precision allowed by the floating-point type used.

Portability has apparently not been an issue for the manufacturers who provided the first elementary functions in Ada. Further progress is to be expected from designers of mathematical software. An initial step was the implementation in Ada of the Cody - Waite software manual, as provided by Whitaker & Eicholtz (1982).

In the above mentioned paper (Kok & Symm, 1984), the acceptance of a standard specification for the basic mathematical functions (like SQRT, LN or LOG, EXP, SIN, COS, ARCTAN) was called a prerequisite for the production of portable, reliable and efficient software for scientific computation (see, for example, Rice, 1983).

In the paper a discussion is given of the following aspects of a generally accepted elementary functions specification :-

- the introduction of the available floating-point types and of user-defined types into the collection of functions (in the sequel it is assumed that an Ada package is given containing the definition of this collection),

- the choice of the basic mathematical functions,

- the specification of each function, including: name, types or

subtypes used for parameters, formal names of parameters and possible defaults, result type,

- hierarchy, if any, of the components of the package of basic mathematical functions,

and, concerning implementation :-

- in what sense may calculations be considered to be portable,

- use of exceptions and other implementation recommendations.

An elaborate discussion can also be found in the Guidelines (Symm et al., 1984).

The adoption of a standard specification of the basic mathematical functions in Ada is currently being pursued by members of the Ada-Europe Numerics Working Group. The (draft) specification to be proposed is derived from the package GENERIC_MATH_FUNCTIONS (Kok & Symm, 1984). For details we refer to the Ada-Europe Numerics WG working paper (Kok, 1987) that contains the proposed Ada specification. Differences concern the place of definition of the mathematical constants and the use of overloading for declarations of elementary functions with an additional (floating-point type) parameter (e.g. with the purpose of scaling the argument of the trigonometric functions).

In deriving the PIA project version from this draft proposed standard specification, another deviation was introduced since otherwise no equivalent of the "**" operator could be made that fully employs the error handling mechanism. Further, the declaration of the exception ARGUMENT_ERROR does not occur in the PIA project version. For this declaration, the library error mechanism has a different solution. It is therefore omitted in NAG_GENERIC_S01BA.

3. PORTABILITY IN THE SPECIFICATION OF ELEMENTARY FUNCTIONS

For a package of elementary functions in Ada, we distinguish between portability of the package specification and portability of the implementation. In this section we discuss the portability of the design of elementary functions, and this affects mainly their specification. The portability of the implementation is discussed in section 5. It is also discussed in Kok & Symm (1984).

Full generality is not obtained by giving function specifications for the available implementation-defined floating-point types. In that case the performance of the implementations is not portable, and the application for user-defined floating-point types brings extra work to the user (choosing the mapping onto the appropriate implementation-defined floating-point type and writing explicit type

conversions). This solution does not use the possibilities of Ada and is therefore not in agreement with a recommendable Ada style.

The Ada style solution is to write library packages which are generic with respect to the real type(s) to be used inside. For the basic mathematical functions, we then have:

```
generic
   type FLOAT_TYPE is digits <>;
package GENERIC_MATH_FUNCTIONS is
   ...
end GENERIC_MATH_FUNCTIONS;
```

This library unit can be used with any user-defined floating-point type but it is recommended that an installation should also provide a standard instance as a library unit for those users who do not want to give further thought to the possibilities which Ada offers here. This would avoid the need for separate instantiations in all dependent packages, possibly yielding many copies of an instance. The standard instantiation would read:

```
with GENERIC_MATH_FUNCTIONS;
package STD_MATH_FUNCTIONS is
   new GENERIC_MATH_FUNCTIONS(FLOAT);
      -- or possibly with LONG_FLOAT.
```

See the Guidelines and Kok & Symm (1984) for the complete specification.

In the generic package above it is assumed, for simplicity, that only one floating-point type is needed by the implementation. It is possible to introduce different floating-point types into the package. We do not pursue this possibility here. Moreover, it has been shown (in Appendix C of Symm et al., 1984) that portable general implementations can be readily designed. Therefore, we prefer to avoid the design of packages with two different generic floating-point type parameters, which are much more difficult to use correctly.

4. ELEMENTARY FUNCTIONS ON TOP OF A LIBRARY ERROR MECHANISM

The project's library error mechanism (NAG_Ada, 1987) can be employed by supplying a generic subprogram parameter FAIL_HANDLER to a library module, and by passing a parameter of the type ERROR_RECORD to every call of a library module. All use is made easy by allowing defaults both in the generic instantiation and in all calls of the library module.

In an instance of the library module the error mechanism is then activated when errors occur. The implementation of a library module will call error mechanism subprograms for storing information about

the error, and by the supplied FAIL_HANDLER some corrective action can be performed.

Since the user has to declare generic instantiations in order to obtain such library modules, we can consider the following possibilities for providing a package of elementary functions on top of the library error mechanism :-

1) the package (which is already a generic package with a generic floating-point type parameter FLOAT_TYPE) is given a second generic parameter FAIL_HANDLER in accordance with the rules given in the library error mechanism documentation,

2) each function in the package is a generic function, with generic parameter FAIL_HANDLER,

3) some functions are generic, other functions for which no errors are expected can be given as 'normal' subprograms.

Disadvantages of the first possibility are that all functions obtain the same FAIL_HANDLER and if different corrective actions are desired for different functions the user has to declare several instantiations of the complete package.

For the second possibility the user has to make instances for each function that is to be used. This is also the case with the third possibility for those functions which are provided as generic subprograms, whereas other functions can be called directly and will therefore have a different specification. Moreover, it is hardly possible to predict that some elementary function will never arrive at an error situation, in particular for the exceptions that the implementation can raise. But it might be decided to let these exceptions propagate without storing information about their first occurrence.

For simplicity and ease of use we have chosen the first possibility, viz. a generic package called NAG_GENERIC_S01BA (see Appendix I for the specification). In this case standard instantiations which use the DEFAULT_HANDLER can be made available in advance. All extra trouble for the user is then removed. The instantiation for the built-in type FLOAT would read:

```
with NAG_GENERIC_S01BA;
package NAG_S01BA_FLOAT is new NAG_GENERIC_S01BA (FLOAT);
```

In Appendix IV we present the source code of an example program for testing the use of the library error mechanism, which is available in the package NAG_P01AA. Appendix V contains the output of this program.

5. THE IMPLEMENTATION OF ELEMENTARY FUNCTIONS

For the portable implementation of the elementary functions in Ada we consider two topics :-
- the portable implementation of sufficiently accurate algorithms for all expected argument domains of the functions,
- the use of the library error mechanism, in agreement with the specification discussed in section 4.

5.a. Portable implementations of sufficient accuracy

For the computation of values of the elementary functions with optimal accuracy, the algorithms have been taken from Cody & Waite (1980), Hemker et al. (1973), and the Guidelines. New algorithms were derived for the inverse hyperbolic functions (see Bergman, 1987).
An additional requirement was suitability to the higher accuracy of the floating-point types aimed at in the Pilot Implementations project. In order to extend the range of application of most functions many approximation functions have been recomputed.

5.b. The handling of errors

For the following error situations an action has been implemented in the bodies of the elementary functions :-

- wrong argument(s)

When the value given for the (floating-point type) argument parameter(s) is outside the range specified for the function concerned, for example when SQRT is called with a negative argument, this is always signalled. Appendix II contains a table in which for all functions the exceptional arguments are specified, together with a list of edge values. An edge value is delivered in a normal return, if a wrong argument value is passed and the supplied FAIL_HANDLER 'clears' the error. For all occurrences of 'wrong argument' the condition of the error will be set to FIRM, which means that the computation will not continue when DEFAULT_HANDLER is the supplied error handler. Further, the error specification will be set to ARGUMENT_ERROR.

In the presence of the DEFAULT_HANDLER the function is exited by raising the exception LIBRARY_EXCEPTION. However, the user can provide a user-defined handler, which simply clears the error if it was 'wrong argument', and then the function will give a normal return delivering the edge value specified. The user-defined handler could contain a statement like:

```
if READ_STATE(FAIL) = FIRM and then
      READ_ERROR(FAIL).CATEGORY = ARGUMENT_ERROR then
   CLEAR_ERROR(FAIL);
end if;
```

- NUMERIC_ERROR raised by the Ada implementation

  Some functions might cause overflow in computations, and then some implementations might choose to raise the STANDARD exception NUMERIC_ERROR. We expect that this will only occur for function calls for which the final mathematical result would also overflow (or sometimes underflow). We have chosen to treat this situation in the same way as for wrong arguments. The error condition will be set to FIRM, and the error specification will be OVERFLOW_ERROR. Appendix II contains a list of edge values which will be delivered if the user-supplied handler clears this error.

  Although the Ada Reference Manual is not very specific about distinguishing NUMERIC_ERROR and CONSTRAINT_ERROR in floating-point computations, we have not implemented an analogous treatment for a raised CONSTRAINT_ERROR. We expect that this exception will normally be raised by the Ada implementation if the user has made an instance of NAG_GENERIC_S01BA with a floating-point type with an extremely narrow additional range constraint. It is recommended that the generic actual parameter (for FLOAT_TYPE) does not have an additional range constraint.

- the occurrence of other predefined exceptions

  All other predefined exceptions are not handled in the implementation of elementary functions.

6. RESULTS

At the first stage of the project we had adopted as the standard for basic mathematical functions the generic package that has been proposed in Symm et al. (1984) and Kok & Symm (1984).

In the project we have departed from this specification in order to employ in the elementary functions a very powerful library error handling mechanism. Thus, the elementary functions can be used in connection with a large scientific library which is based on this library error mechanism. Moreover, the final version of the package specification for the PIA project is now derived from the current draft specification to be proposed for standardisation.

The (generic) package specification has been designed such that the components of the two package specifications appear identical when defaults are used for all parameters related to the library error mechanism, except for the name of the package and for the declaration of the exception ARGUMENT_ERROR.

It is shown that the package of elementary functions can without difficulty be designed and implemented on top of the library error handling core module which is another result of the Pilot

Implementations project. Anticipating the adoption of a standard for elementary functions in Ada, the implementation of NAG_GENERIC_S01BA is also based upon the generic package GENERIC_MATH_FUNCTIONS which has the same purpose with the exception of the use of the advanced library error handling mechanism. This project result actually shows the implementability in Ada of both the draft proposed package GENERIC_MATH_FUNCTIONS and the PIA project package.

Finally, we remark that it has been shown that similar specifications can easily be used also for use in a real-time environment when the implementations (the functions bodies) contain tasks, although a replacement of functions by tasks might be considered (Kok, 1985).

7. DOCUMENTATION

Complete documentation of the generic package, all subprograms comprised in it, and of the standard library instance (i.e. for the library type NAG_A01AA_REAL.REAL) is available in the NAG_Ada Library Document S01BA (NAG_Ada, 1987). This documentation also contains an example program which uses all functions both in calls with one and calls with two floating-point type arguments.
Chapter S01 further contains the NAG_Ada Library Document S01AA providing the mathematical constants.

8. TESTING

Testing of library software of Ada is needed for judging the choice of algorithms, for certifying that the chosen algorithms have been implemented in Ada correctly, for checking that generic actual parameters of generic instances satisfy the assumptions made by the generic module, and for verifying the portability of the implementation.

With the implementation of the elementary functions package testing can be performed in the following directions:-

- correct argument domain, and signalling of excess of bounds,

- function result range (mathematically), if specified,

- accuracy of results:
  = compared with known values,
  = substitution in mathematical relationships with simple answers (and for which a stable floating-point computation is possible),
  = compared with results of other implementations on the same machine,
  = compared with results obtained on different machines,

- performance of the use made of the library error handling mechanism,
- re-usability (different generic actual parameters),
- portability (compilation and execution on different machines),
- efficiency.

Testing of the consistency of generic actual parameters has been omitted, since the certification of the appropriateness of the generic actual floating-point type can quite well be established by the package itself.

Since the aim of this part of the project was to obtain portable code IN Ada, efficiency has not been investigated. Care has been taken, however, that algorithms were implemented for all expected floating-point type characteristics in such a way that the sufficiently accurate results of evaluations are obtained as fast as possible.

The result of the PIA project concerning the testing of the elementary functions implementation is a portable implementation in Ada of the tests given in Cody-Waite (1980), extended for the inverse hyperbolic functions. The tests comprise:-

- computing the maximum magnitude and the root mean square of the relative errors of a large set of randomly chosen arguments in specified intervals,
- substitution in easy-to-compute mathematical identities,
- testing of near-boundary argument values,
- testing of the behaviour for exceptional arguments and, subsequently, of the error handling mechanism.

On the Ada machine in operation at the CWI all these tests were performed for package instances with the built-in types FLOAT and LONG_FLOAT. Furthermore, all tests with the exception of testing the error handling mechanism were also performed for instances with the user-defined floating-point types

digits 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 and 14.

In all cases, the reported accuracy and the behaviour for mathematical relationships, near-boundary and excess argument values was completely satisfactory.

Finally, a comparison was made of SQRT, LOG, EXP, SIN, TAN, ARCSIN and ARCTAN with comparable facilities provided by the Ada system in operation at the CWI. The differences in accuracy that were found were always within tolerance.

ACKNOWLEDGEMENTS

The work reported in this paper was performed by the author in co-operation with Martin Bergman and Dik T. Winter of the CWI. We acknowledge the helpful comments received from colleagues in the PIA Project, from the CEC Representatives involved and from the Reviewers appointed by the Commission of the European Communities.

REFERENCES

ANSI/MIL-STD 1815 A. Reference manual for the Ada programming language, January 1983
(ISO/8652-1987 Programming languages - Ada, March 1987).

Bergman, M. Implementation of elementary functions in Ada, CWI Report NM-R8709, 1987.

Cody, W.J. and Waite, W. Software manual for the elementary functions, Prentice-Hall, New Jersey, 1980.

Ford, B., Kok, J. & Rogers, M.W. Scientific Ada, Cambridge University Press, Cambridge U.K., 1986.

Hemker, P.W., Hoffmann, W., Kampen, S.P.N. van, Oudshoorn, H.L. & Winter, D.T. Single and double-length computation of elementary functions, NW 7/73, Mathematisch Centrum, Amsterdam, 1973.

Kok, J. Two Ada mathematical functions packages for use in real time, CWI Report NM-R8512, June 1985.

Kok, J. Proposal for standard mathematical packages in Ada, Ada-Europe Numerics Working Group working paper A-ENWG/13.5, 1987.

Kok, J. and Symm, G.T. A proposal for standard basic functions in Ada, Ada Letters, Vol. IV.3, Nov/Dec 1984, 44-52.

NAG Ada Library Manual, NAG Ltd, Oxford, 1987.

Rice, J.R. Remarks on software components and packages in Ada. ACM SIGSOFT Software Engineering Notes, 1983, 8(2), 9-10.

Symm, G.T., Wichmann, B.A., Kok, J., and Winter, D.T. Guidelines for the design of large modular scientific libraries in Ada,
NPL Report DITC 37/84 and CWI Report NM-N8401, March 1984,
also Chapter 10 in: Ford et al., 1986.

Whitaker, W.A. and Eicholtz, T.C. An Ada implementation of the Cody-Waite "Software manual for the elementary functions", US Air Force, 1982.

Appendix I: SPECIFICATION OF THE PACKAGE OF ELEMENTARY FUNCTIONS

```
  -- Standard specifications with defaults.
  --================================================================
  ----------------------------------------------------------
  --                                                        --
  --  Copyright 1987 PIA Consortium (all rights reserved)   --
  --                                                        --
  ----------------------------------------------------------
  --                                                        --
  with NAG_P01AA;
  use NAG_P01AA;
  generic

  -- Types

     type FLOAT_TYPE is digits <>;

  -- Subprograms

     with procedure FAIL_HANDLER(FAIL : ERROR_RECORD)
        is DEFAULT_HANDLER;

  package NAG_GENERIC_S01BA is

  --       Subprograms

     function SQRT (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function LOG (X, BASE : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function LOG (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function EXP (X, BASE : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function EXP (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function "**" (X, Y : FLOAT_TYPE) return FLOAT_TYPE;
     function SIN (X, CYCLE : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function SIN (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function COS (X, CYCLE : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function COS (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function TAN (X, CYCLE : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
     function TAN (X : FLOAT_TYPE;
                    FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
```

```
   function COT (X, CYCLE : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function COT (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCSIN (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCSIN (X, CYCLE : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOS (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOS (X, CYCLE : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCTAN (X : FLOAT_TYPE;
            Y : FLOAT_TYPE := 1.0;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCTAN (X : FLOAT_TYPE;
            Y : FLOAT_TYPE := 1.0;
            CYCLE : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOT (X : FLOAT_TYPE;
            Y : FLOAT_TYPE := 1.0;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOT (X : FLOAT_TYPE;
            Y : FLOAT_TYPE := 1.0;
            CYCLE : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function SINH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function COSH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function TANH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function COTH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCSINH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOSH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCTANH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;
   function ARCCOTH (X : FLOAT_TYPE;
            FAIL : ERROR_RECORD := DEFAULT_RECORD) return FLOAT_TYPE;

end NAG_GENERIC_S01BA;
--=============================================================================
```

Appendix II: EDGE VALUES FOR WRONG FUNCTION ARGUMENTS

Exceptional arguments and edge values are given by :-

Function	Edge value of result with overflow	wrong argument	error in argument
SQRT	-.-	0.0	X < 0.0
LOG	-.-	sign*REAL'LARGE	X <= 0.0
		sign*REAL'SMALL	BASE <= 0.0
		REAL'LARGE	BASE = 1.0
EXP	REAL'LARGE		(for X > 0.0)
	REAL'SMALL		(for X < 0.0)
		REAL'SMALL	BASE < 0.0 or
			BASE = 0.0 and X <= 0.0
"**"	REAL'LARGE		(for Y > 0.0)
	REAL'SMALL		(for Y < 0.0)
		REAL'SMALL	X < 0.0 or
			X = 0.0 and Y <= 0.0
SIN	-.-	0.0	CYCLE = 0.0
COS	-.-	0.0	CYCLE = 0.0
TAN	REAL'LARGE	0.0	CYCLE = 0.0
COT	REAL'LARGE	0.0	CYCLE = 0.0
		REAL'LARGE	X = 0.0
ARCSIN	-.-	-CYCLE/4.0	X < - 1.0
		CYCLE/4.0	X > 1.0
		0.0	CYCLE = 0.0
ARCCOS	-.-	CYCLE/2.0	X < - 1.0
		0.0	X > 1.0
		0.0	CYCLE = 0.0
ARCTAN	-.-	CYCLE/8.0	X = 0.0 and Y = 0.0
		0.0	CYCLE = 0.0
ARCCOT	-.-	CYCLE/8.0	X = 0.0 and Y = 0.0
		0.0	CYCLE = 0.0
SINH	sign*REAL'LARGE		
COSH	REAL'LARGE		
TANH	-.-		
COTH	-.-	REAL'LARGE	X = 0.0
ARCSINH	-.-		
ARCOSH	-.-	0.0	X < 1.0
ARCTANH	s*REAL'LARGE	sign*REAL'LARGE	abs X >= 1.0
ARCCOTH	s*REAL'LARGE	sign*REAL'LARGE	abs X <= 1.0

Note: Complicated (conditional) expressions for the signs of some edge values are omitted, and the identifier REAL refers to the generic type parameter FLOAT_TYPE. Further, edge values given are rather arbitrary when the function has no (unique) limit.

Appendix III: IMPLEMENTATIONS : NAG_GENERIC_S01BA and LOG.

```
-- NAG_GENERIC_S01BA Body :
with NAG_P01AC,
     NAG_S01AA, -- Used at least for temporary implementation.
     GENERIC_MATH_FUNCTIONS;
package body NAG_GENERIC_S01BA is
  package MATH_FUNCTIONS_FLOAT_TYPE is new GENERIC_MATH_FUNCTIONS
       (FLOAT_TYPE);
-------------------------------------------------------------------------
-- Auxiliary procedure WRONG_ARGUMENT using NAG_P01AA, NAG_P01AC  --
-------------------------------------------------------------------------
  procedure WRONG_ARGUMENT (STATUS   : in ERROR_CONDITION;
                            NUMBER   : in INTEGER;
                            LIB_UNIT : in STRING;
                            FAIL     : in ERROR_RECORD) is
  begin
    NEW_ERROR
       (STATE    => STATUS,
        NUMBER   => NUMBER,
        NAME     => LIB_UNIT,
        MESSAGE  => NAG_P01AC.ARGUMENT_ERROR,
        CATEGORY => ARGUMENT_ERROR,
        FAIL     => FAIL);
    FAIL_HANDLER (FAIL);
  end WRONG_ARGUMENT;
-------------------------------------------------------------------------
-- Auxiliary procedure CATCH_NUMERIC_ERROR using NAG_P01AA,NAG_P01AC
-------------------------------------------------------------------------
  procedure CATCH_NUMERIC_ERROR (STATUS   : in ERROR_CONDITION;
                                 NUMBER   : in INTEGER;
                                 LIB_UNIT : in STRING;
                                 FAIL     : in ERROR_RECORD) is
  begin
    NEW_ERROR
       (STATE    => STATUS,
        NUMBER   => NUMBER,
        NAME     => LIB_UNIT,
        MESSAGE  => NAG_P01AC.OVERFLOW_ERROR,
        CATEGORY => OVERFLOW_ERROR,
        FAIL     => FAIL);
    FAIL_HANDLER (FAIL);
  end CATCH_NUMERIC_ERROR;
-------------------------------------------------------------------------
-- Body stubs of the basic mathematical functions.                    --
-------------------------------------------------------------------------
  function SQRT (X    : FLOAT_TYPE;
                 FAIL : ERROR_RECORD := DEFAULT_RECORD)
                  return FLOAT_TYPE is separate;

  -- Temporary inaccurate implementation for default.
```

```
   function LOG (X    : FLOAT_TYPE;
                 FAIL : ERROR_RECORD := DEFAULT_RECORD)
                  return FLOAT_TYPE is
   begin
     return LOG (X, BASE => NAG_S01AA.NATURAL_E, FAIL => FAIL);
   end LOG;

   function LOG (X, BASE : FLOAT_TYPE;
                 FAIL    : ERROR_RECORD := DEFAULT_RECORD)
                  return FLOAT_TYPE is separate;

   -- etc.

end NAG_GENERIC_S01BA;
--============================================================================


-- Template for an elementary function, here LOG, in which the
-- actual computation is performed by GENERIC_MATH_FUNCTIONS.LOG
--
separate ( NAG_GENERIC_S01BA )
function LOG (X, BASE : FLOAT_TYPE;
              FAIL : ERROR_RECORD := DEFAULT_RECORD)
               return FLOAT_TYPE is
begin
  CHECK_RECORD (FAIL);
  return MATH_FUNCTIONS_FLOAT_TYPE.LOG (X, BASE);
exception
  when MATH_FUNCTIONS_FLOAT_TYPE.ARGUMENT_ERROR =>
    if X <= 0.0 then
      WRONG_ARGUMENT (FIRM, 1, "LOG", FAIL);
      case READ_STATE (FAIL) is
        when HARD =>  raise LIBRARY_EXCEPTION;
        when others =>
          if BASE > 1.0 then
            return -FLOAT_TYPE'LARGE;
          else
            return FLOAT_TYPE'LARGE;
          end if;
      end case;
    end if;
    if BASE <= 0.0 then
      WRONG_ARGUMENT (FIRM, 2, "LOG", FAIL);
      case READ_STATE (FAIL) is
        when HARD =>  raise LIBRARY_EXCEPTION;
        when others =>
          if X > 1.0 then
            return -FLOAT_TYPE'SMALL;
          else
            return FLOAT_TYPE'SMALL;
          end if;
```

```
      end case;
    end if;
    if BASE = 1.0 then
      WRONG_ARGUMENT (FIRM, 3, "LOG", FAIL);
      case READ_STATE (FAIL) is
        when HARD =>  raise LIBRARY_EXCEPTION;
        when others =>
          return FLOAT_TYPE'LARGE;
      end case;
    end if;
  when NUMERIC_ERROR =>  -- system_generated
    CATCH_NUMERIC_ERROR (HARD, 4, "LOG", FAIL);
    raise LIBRARY_EXCEPTION; -- ignore user-defined handler
end LOG;
```

Appendix IV: ADA EXAMPLE PROGRAM FOR USE OF NAG_GENERIC_S01BA

```
-- 860107, Example and test for :
-- J. Kok, 851203 (version 861201), error handling according to :-
--    NAG_Ada Manual, Chapter P01 (1987).
--

-- Example program for instance of NAG_GENERIC_S01BA (NAG_S01BA_REAL)
-- (which uses NAG_P01AA, NAG_P01AC, and NAG_GENERIC_S00AA)
--
with TEXT_IO,
     INTEGER_IO,       -- standard instance
     FLOAT_IO,         -- standard instance
     NAG_S01AA,        -- mathematical constants
     NAG_S01BA_REAL;   -- standard library instance
use TEXT_IO, INTEGER_IO, FLOAT_IO, NAG_S01AA, NAG_S01BA_REAL;
with NAG_P01AA, NAG_P01AB;
use NAG_P01AA, NAG_P01AB;
procedure LOG_EXAMPLE is

  REC1, REC2, REC3 : ERROR_RECORD;
  FX               : FLOAT;


  procedure TEST (TITLE : in STRING;
                  ARG1  : in FLOAT;
                  ARG2  : in FLOAT := NATURAL_E;
                  FAIL  : in ERROR_RECORD) is
  begin
    PUT_LINE (READ_CALLING_UNIT (FAIL) & " -- " & TITLE &
       ", args are:");
    PUT (ARG1, 3, 5);
    PUT (ARG2, 3, 5);
    NEW_LINE;
    FX := LOG (ARG1, ARG2, FAIL);
    PUT ("Result is:");
    PUT (FX, 4, 5);
    PUT_ERROR ( FAIL );
    NEW_LINE;

  exception
    when others =>
      PUT_ERROR ( FAIL );
      NEW_LINE;
  end TEST;

begin
  UNSAFE; -- for sequential processing
  NEW_RECORD (REC1, "Test 1");
  NEW_RECORD (REC2, "Test 2");
```

```
    NEW_RECORD (REC3, "Test 3");
    TEST (" good argument", 0.5, 2.0, REC1);

    TEST (" base too small", 0.5, -0.5, REC2);

    TEST (" base = one", 10.0, 1.0, REC3);

    TEST (" base = one, previous error not reset",
          10.0, 1.0, REC3);

    PUT_LINE ("End of tests.");

end LOG_EXAMPLE;

pragma MAIN;
```

Appendix V: RESULTS OF EXAMPLE PROGRAM

```
Test 1 --  good argument, args are:
 5.00000E-01  2.00000E+00
Result is:  -1.00000E+00
No Library Error Recorded - Possible Predefined or User Exception ?

Test 2 --  base too small, args are:
 5.00000E-01 -5.00000E-01

HARD error 2 in library unit LOG
ARGUMENT_ERROR

Test 3 --  base = one, args are:
 1.00000E+01  1.00000E+00

HARD error 3 in library unit LOG
ARGUMENT_ERROR

Test 3 --  base = one, previous error not reset, args are:
 1.00000E+01  1.00000E+00

HARD error 3 in library unit LOG
RECORD_NOT_CLEAR

End of tests.
```