ISO/IEC JTC1/SC22/WG5 N1209

A SPECIFIC PROPOSAL FOR INTERVAL ARITHMETIC IN FORTRAN

March 20, 1996

compiled by

R. Baker Kearfott

as a result of discussions in the group consisting of

Keith Bierman
George Corliss
David Hough
Andrew Pitonyak
Michael Schulte
William Walster
Wolfgang Walter

CONTENTS

==

I. INTRODUCTION

This document is to propose standard syntax and requirements for interval computations in Fortran, and to give occasional guidelines for implementation. These requirements are based upon precedents, as well as thorough discussion of various points.

II. GENERAL PRINCIPLES

Containment: Every interval result must CONTAIN the exact mathematical range of the corresponding point operation or function evaluation.

> Note: Containment is crucial for verification and validation computations to be rigorous, and is easy to achieve using floating point arithmetic and directed rounding, as defined by the IEEE 754 standard, or (less accurate) simulations thereof.

Accuracy: There is no accuracy requirement.

> Note: The lack of an accuracy requirement facilitates universal implementation of the standard syntax.
>
> Note: Ideally, results of the elementary operations and intrinsic functions should have as small a width as is mathematically possible subject to the condition that the result contain the exact mathematical range. This is not difficult for the four basic operations "+", "-", "*", and "/", if IEEE 754 arithmetic is available. For the standard functions, a more realistic goal is that the lower bound be at most 1ULP (unit in the last place) less than the ideal lower bound, and that the upper bound be at most 1ULP more than the ideal upper bound.

Speed: There is no speed requirement.

> Note: It is reasonable to expect machine-specific implementations to come within a factor of 5, or perhaps within a factor of 2, of the speed of point arithmetic. This is particularly true if IEEE 754 is available and if the floating point standard function library was designed to produce results with known relative accuracy.

Intrinsic data type: The proposed standard requires an intrinsic interval data type to implement interval arithmetic in Fortran.

> Note: Some, but not all, aspects of the standard can be implemented in a module. In particular, the interval edit descriptors for interval I/O (VE, VF, and VG formats, as well as extensions of the standard formats to interval computations) cannot be implemented within a module, although subroutine calls can be used to implement interval I/O. The representation of interval constants within Fortran executable statements, e.g. representing the interval [1,2] as (<1,2>) on the right side of an assignment statement, cannot be done with a module. Also, type parameters for the interval data type cannot be implemented with a module. Finally, there is no way to define the natural precedence of interval operators within a module.

In addition to the containment requirement, the following items are required:

* an INTERVAL numeric data type, obeying the same syntax as the other numeric data types,

* several new infix operators and intrinsics,

* INTERVAL versions of all Fortran 90 intrinsics that accept REAL data,

* natural extensions to Fortran standard I/O to include intervals.

> Note: A complex interval data type is neither required nor prohibited.

> Note: Various forms of extended interval arithmetic are neither prohibited nor required. There are at least three different extended interval arithmetics (Kahan arithmetic, Kaucher arithmetic, and Markov arithmetic), all useful but designed for different purposes.

III. THE INTERVAL DATA TYPE AND INTERVAL INTRINSIC FUNCTIONS

A. The Data Type and Basic Operations

Name and structure: The INTERVAL type is a numeric type. Its values are closed and bounded real interval which are defined by an ordered pair of real values. The first real value is the lower bound (or infimum), the second real value is the upper bound (or supremum). All real numbers between and including these two bounds (or endpoints) are said to be elements of the interval.

The two storage units of an interval type are each of the same real type available on the processor. The individual storage units are accessible through the functions INF and SUP defined below.

Arithmetic operations: The four basic operations +, -, *, are defined to contain the ranges of the corresponding operations on real numbers. Specifically, let X = [xl,xu] an Y = [yl,yu] be intervals, where xl represents the lower bound of X, xu represents the upper bound of X, yl represents the lower bound of Y, and yu represents the upper bound of Y. Then:

X + Y shall contain the exact value [xl + yl, xu + yu],

X - Y shall contain the exact value [xl - yu, xu - yl],

X * Y shall contain the exact value
```
      [min{xl*yl, xl*yu, xu*yl, xu*yu}, max{xl*yl, xl*yu, xu*yl, xu*yu}]
```

1 / X shall contain the exact value

```
        | [1/xu, 1/xl]   xu < 0 or xl > 0
        |
        | [-inf, inf]    otherwise
```

where "inf" is the abbreviation for the symbols IEEE_NEGATIVE_INF and IEEE_POSITIVE_INF stipulated below.

X / Y shall contain the exact value
```
      [min{xl/yu, xl/yl, xu/yu, xu/yl}, max{xl/yu, xl/yl, xu/yu, xu/yl}]

      if yu < 0 or yl > 0

      and shall be equal to [-inf, inf] otherwise.
```

Note: The second case of interval division may be replaced by sharper definitions on particular processors in a separate extended interval arithmetic.

Note: Using floating point arithmetic, the operations on the right-hand sides may first be computed, then the lower bound may be rounded down to a number known to be less than or equal to the exact mathematical result, and the upper bound may be rounded up to a number known to be greater than or equal to the exact mathematical result. If a processor provides directed roundings upwards (towards plus infinity) and downwards (towards minus infinity), then the operation and the rounding can be performed in one step, e.g. if the processor conforms to the IEEE 754 standard. The excess interval width caused by this outward rounding is called ROUNDOUT ERROR.

Note: There is an alternate implementation of interval multiplication that also gives the range of the real operator "*" over the intervals X and Y. This alternative involves nine cases determined by the algebraic signs of the endpoints of X and Y; see page 12 of R. E. Moore, "Methods and Applications of Interval Computations," SIAM, Philadelphia, 1979. The average number of multiplications required for this alternative is less than above, but one or more comparisons are required. Implemented in software, the relative efficiencies of the alternative above and the nine-case alternative are architecture-dependent, although the nine-case alternative is often preferred in low-level implementations designed for efficiency.

Note: The only processor requirement is that the computed intervals contain the exact mathematical range of the corresponding point operations. In an ideal implementation (not required), the result of the operations is the smallest-width machine interval that contains the exact mathematical range.

Note: IEEE arithmetic helps with outward roundings. For example take

```
[xl,xu] + [yl,yu] = [xl+yl,xu+yu]
```

in exact interval arithmetic. The IEEE 754 standard defines a downwardly rounded operation as producing the same result as would be obtained by computing the exact result, then rounding it to the nearest floating point number less than or equal to the exact result, and an upwardly rounded operation as producing the same result as would be obtained by computing the exact result, then rounding it to the nearest floating point number greater than or equal to the exact result. Thus, if the result xl+yl is rounded down and xu+yu is rounded up according to the IEEE specifications, an ideal interval addition results.

Infinities: Two (case-insensitive) symbols, IEEE_NEGATIVE_INF and IEEE_POSITIVE_INF, shall also be defined.

Note: IEEE_NEGATIVE_INF and IEEE_POSITIVE_INF correspond to negative infinity and positive infinity in IEEE arithmetic, but shall be defined on all processors. The symbol IEEE_NEGATIVE_INF represents something less than all floating point numbers, and IEEE_POSITIVE_INF represents something greater than all floating point numbers. Intervals with one or both endpoints equal to these symbols shall be allowed, and arithmetic on them is defined, consistently with IEEE arithmetic.

The empty set: The interval (<IEEE_POSITIVE_INF,IEEE_NEGATIVE_INF>) shall represent the empty set.

Note: Intervals in which the upper endpoint is less than the lower endpoint are non-standard. However, various useful non-standard extensions can be based on such representations.

Mixed mode operations: mixed-mode INTERVAL/REAL and mixed mode INTERVAL/INTEGER operations shall be defined. The result of such a mixed-mode operation shall be the same as if the other data type (INTEGER or REAL) were first converted to an INTERVAL that contains the mathematical interpretation of the original data type.

Note: Mixed-mode INTERVAL/COMPLEX is not defined.

No implicit conversion from interval: Implicit conversion from interval to other data types shall not be allowed.

Note: The functions INF and SUP defined below may be used to convert an INTERVAL to another data type. Similarly, MID may also be viewed as producing a real approximation to an interval, while INTERVAL converts from real to interval.

Implicit conversion to interval: Implicit conversion to interval shall be possible. The result of an implicit conversion to interval shall contain the mathematical interpretation of the original data type.

Type parameters: The INTERVAL data type shall admit one or more type parameters. Each type parameter shall be equal to the corresponding REAL type parameter.

Note: The default INTERVAL type should correspond to a REAL type with at least 64 bits. ("DOUBLE PRECISION" on many machines). It is the consensus of experts that 32-bit interval arithmetic is of limited use.

B. New Infix Operators

The following infix operators shall be a part of standard interval support.

Syntax	function
Z = X.IS.Y	Z <-- intersection of X and Y, that is, [max{xl,yl},min{xu,yu}] if max{xl,yl} < = min{xu,yu} and [inf,-inf] otherwise.
Z = X.CH.Y	Z <-- [min{xl,yl}, max{xu,yu}] ("interval hull" of X and Y. The mnemonic is "convex hull")
X.SB.Y	.TRUE. if X is a subset of Y (i.e. if xl >= yl .AND. xu <= yu)
X.PSB.y	.TRUE. if X is a proper subset of Y (i.e. if X.SB.Y .AND. (xl > yl .OR. xu < yu)
X.SP.Y	.TRUE. if and only if Y.SB.X is true (i.e. if xl <= yl .AND. xu >= yu)
X.PSP.Y	.TRUE. if and only if Y.PSB.X is true (i.e. if Y.SB.X .AND. (yl > xl .OR. yu < xu)
X.DJ.Y	.TRUE. if X and Y are disjoint sets (i.e. if xl > yu or xu < yl)
R.IN.X	.TRUE. if the REAL value R is contained in the interval X (i.e. if xl <= R <= xu)

Note: Intervals are viewed as closed intervals, so, if R.IN.X, then R may be equal to one of the endpoints of X.

C. Interval Versions of Relational Operators

The following relational operators shall be extended to interval operations, in the "certainly true" sense. That is, the result is .TRUE. if and only if it is true for each pair of real values taken from the corresponding interval values.

Syntax	function
X.LT.Y	.TRUE. if xu < yl

X.GT.Y	.TRUE. if xl > yu
X.LE.Y	.TRUE. if xu <= yl
X.GE.Y	.TRUE. if xl >= yu

As with non-interval data types in the Fortran standard, the newer symbols "<", ">", "<=",and ">=" shall be interchangeable with ".LT.", ".GT.", ".LE.", and ".GE.", respectively.

Another set of relational operators, the POSSIBLY TRUE relationals, shall be defined as follows.

Syntax	function	
X.PLT.Y	.TRUE. if xl < yu	(i.e. if .NOT.(X.GE.Y))
X.PGT.Y	.TRUE. if xu > yl	(i.e. if .NOT.(X.LE.Y))
X.PLE.Y	.TRUE. if xl <= yu	(i.e. if .NOT.(X.GT.Y))
X.PGE.Y	.TRUE. if xu >= yl	(i.e. if .NOT.(X.LT.Y))

Finally, equality and inequality of intervals are defined by viewing the intervals as sets.

Syntax	function
X.EQ.Y	.TRUE. if xl=yl and xu=yu
X.NE.Y	.TRUE. if .NOT. (X.EQ.Y)

Note: "/=" and "==" shall be interchangeable with ".NE." and ".EQ.", respectively.

D. Special Interval Functions

The following utility functions shall be provided for conversion from INTERVAL to REAL, etc.

```
  Syntax      function                  attainable accuracy
  ______      ________                  ___________________

R = INF(X)    Lower bound of X          (the value in the lower
                                         storage unit of the interval
                                         datum X)

R = SUP(X)    Upper bound of X          (the value in the upper
                                         storage unit of the interval
                                         datum X)

R = MID(X)    Midpoint of X             (a floating point approximation,
                                         always greater than or equal to
                                         the value returned by INF and
                                         less than or equal to the value
                                         returned by SUP)

R = WID(X)    R <-- xu - xl             (the value shall be rounded up,
              "Width"                    to be greater than or equal to
                                         the actual value)

R = MAG(X)    R <-- max { |xl|, |xu| }
              "Magnitude"

                   | min { |xl|, |xu| } if .NOT.(0.IN.X),
R = MIG(X)    R <--|
                   | 0 otherwise.

              "Mignitude"


                    |
Z = ABS(X)    Z <-- | [min{|x|}, max{|x|}]
                    |  x.IN.X    x.IN.X


              Range of absolute value

Z = MAX(X,Y)  Z <-- [max {xl,yl}, max {xu,yu}]

              Range of maximum
              MAX shall be extended analogously
              for more than two
              arguments.

Z = MIN(X,Y)  Z <-- [min {xl,yl}, min {xu,yu}]
```

```
              Range of minimum
              MIN shall be extended analogously
              for more than two
              arguments.


N = NDIGITS(X) Number of leading decimal digits that are the same in
              xl and xu. n digits shall be counted as the same if
              rounding xl to the nearest decimal number with n
              significant digits gives the same result as rounding
              xu to the nearest decimal number with n significant
              digits.

Z = INTERVAL(R,S) Z <-- [R,S]           (see below)
Z = INTERVAL(R)   Z <-- [R,R]
```

The conversion function INTERVAL shall be an enclosure for the specified interval, with an ideal enclosure equal to a machine interval of minimum width that contains the exact mathematical interval in the specification.

Note: On many machines, INF, SUP, MAG, MIG, ABS, MAX, and MIN can be exact, if the target is of a type that corresponds to the input. This is because these functions merely involve storing one of the endpoints of the interval into the target variable. Similarly, the conversion function INTERVAL can be exact on such machines if it specifies conversion from REAL data of corresponding type.

All of these functions except MAX, MIN, and NDIGITS shall be elemental functions.

Note regarding conversion of decimal constants: If R or S are decimal constants, then a conversion error can occur before the conversion to INTERVAL. For this reason, such quantities should be input as interval constants (see I/O below), rather than with the function INTERVAL. For example, in the statement

```
Z = INTERVAL( 0.500000000000000000000000000123454321),
```

the constant 0.500000000000000000000000000123454321 will first be converted to an internal representation equal to 0.5, then the internal representation of the stored interval is [0.5,0.5], an interval that does not contain the interval constant. However, if an interval constant (defined at the beginning of the I/O section below) is used, the statement

```
Z = (< 0.500000000000000000000000000123454321>)
```

causes the internal representation for Z to contain the value 0.50000000000000000000000000123454321.

Note regarding NDIGITS: For example, if X = [0.1996,0.2004], then three leading decimal digits of this function are the same, and NDIGITS(X) is equal to 3. This is because, if .1996 and .2004 are each rounded to the nearest decimal number with three significant digits, they both round to .200, yet they round to different four-digit decimal numbers.

Note: three interval functions, MAG, MIG, and ABS, correspond to the point intrinsic ABS. The specification of ABS is as the range of the absolute value function, consistent with the general principle that the results of interval functions shall contain the ranges of corresponding point intrinsics. Although "MAG(X)" is written |X| in much of the interval literature, it is more natural in various applications to have ABS(X) denote the range of the absolute value function.

Note: The function WID(X) shall be upwardly rounded, since it often appears in convergence criteria of the form WID(X) < EPS. The criterion is certain to be satisfied if the computed value WID(X), used in the comparison, is greater than or equal to the exact value.

E. Interval versions of the intrinsic functions

General requirements are:

* All Fortran intrinsic functions that accept REAL data shall also accept INTERVAL data.

* All functions shall return enclosures of the range.

* Those generic intrinsics that are REAL elemental functions shall also operate as elemental functions with INTERVAL vector data.

Note: The sharpness of the enclosures is not specified, but an ideal enclosure should be the smallest-width interval with machine numbers as endpoints that contains the actual range. Thus, the only accuracy requirement for interval versions of the intrinsics mandates that they contain the range of the corresponding mathematical function over the set of interval arguments. (In some cases, such as when the argument to the intrinsic contains a pole of the function, the range will be the interval [-inf,inf].)

IV. INTERVAL I/O

The following are specified:

* the form of INTERVAL constants;

* conversion of INTERVAL constants to internal storage;

* four formats for input and output of INTERVAL values.

The underlying principle is the same as with the four basic operations and the interval intrinsic functions: the interval result shall contain the exact mathematical result. Specifically:

* On input, the stored interval shall contain the interval represented by the character input string.

* On output, the printed interval shall contain the internally represented interval.

A. Interval Constants

Both where literal constants are admitted in a program and as input or output data, INTERVAL's shall be represented by a single REAL or INTEGER or a pair of REAL's, INTEGER's, or combinations thereof, beginning with "(<", separated by "," if there are two numbers, and ending in ">)". For example

(<1,2>), (<1E0,3>), (<1>), and (<.1234D5>)

are all valid INTERVAL constants. An INTERVAL constant specified by a single number is the same as an INTERVAL constant specified by two numbers, both of whose endpoints are equal to the single number. When such a decimal constant is converted to its internal representation, the internal representation shall contain the decimal constant, regardless of how many digits are specified by the decimal constant. For example, upon execution of the statement:

```
X = (<0.31415926535897932384626433832795028D+01>)
```

the interval X shall contain the smallest-width machine interval that contains the number 3.1415926535897932384626433832795028.

Note: Thus, on machines in which the interval data type X appearing in the example above contained components with accuracy that corresponded to less than 35 decimal digits, the interval X would contain the mathematical number PI.

Note: In contrast, the statement

```
X = 0.31415926535897932384626433832795028D+01
```

is allowed, since implicit conversion is allowed. However, the compiler can first convert the REAL decimal constant to a valid REAL that may correspond to fewer digits than the original representation. When this REAL is rounded into an interval, the resulting interval does not necessarily contain PI.
As a simpler example of this phenomenon, take the assignment statement

```
X = 0.50000000000000000000000000000123454321
```

If the internal representation of a real only corresponded to 16 decimal digits, then the right-hand side may first be rounded to the binary equivalent of

```
0.5000000000000000.
```

The interval X would then be the binary equivalent of

```
(<0.5000000000000000,0.5000000000000000>),
```

and X would not contain the original right-hand-side. To guarantee X contains the actual right-hand side, the statement

```
X = (<0.50000000000000000000000000000123454321>)
```

should be used.

B. The Interval VF Edit Descriptor

The VF edit descriptor is of the form VF<w>.<d>. Here, <w> is meant to be the width of each of the two numeric fields of the output or input, and <d> is the number of units to the right of the decimal place in each of the two output fields. Formally, an output field for a VF<w>.<d> edit descriptor is of the form <vf-output-field>, where

```
1) vf-output-field is (<f-output-field_sub1, f-output-field_sub2>)

2) f-output-field  is the output field for the F<w>.<d>
                      format, where <w> and <d> are the
                      values specified in the VF<w>.<d>
                      edit descriptor.
```

The value corresponding to f-output-field_sub1 shall be less than or equal to the exact lower bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation. The value corresponding to f-output-field_sub2 shall be greater than or equal to the exact upper bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation.

It shall be possible to use the VF edit descriptor to output REAL data. In that case, the value corresponding to f-output-field_sub1 shall be less than or equal to the exact value of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation. The value of f-output-field_sub2 shall be greater than or equal to the exact upper bound of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation.

An input field for a VF<w>.<d> edit descriptor is of the form vf-input-field, where

```
3) vf-input-field is f-input-field
                  or (< f-input-field >)
                  or (< f-input-field_sub1, f-input-field_sub2 >)

4) f-input-field  is  a valid input field for the F<w>.<d>
                        format, where <w> and <d> are the
                        values specified in the VF<w>.<d>
                        edit descriptor.
```

If vf-input-field is of the form (< f-input-field_sub1, f-input-field_sub2 >), then f-input-field_sub1 represents the lower bound and the f-input-field_sub2 represents the upper bound. In this case, the lower bound of the internal representation of the variable in the corresponding input item list shall be less than or equal to f-input-field_sub1, and the upper bound of the variable shall be greater than or equal to f-input-field_sub2, regardless of the number of digits in the field and number of digits in the internal representation.

If vf-input-field is of the form f-input-field or (< f-input-field >), then the internal representation of the corresponding variable in the input item list shall have lower bound that is less than or equal to the value represented by f-input-field, and shall have upper bound that is greater than or equal to f-input-field, regardless of the number of digits in the field and number of digits in the internal representation.

In ALL interval input fields, blanks between an initial "(<" and the first numerical fields, blanks to the left and right of a separating ",", and blanks to the left of a closing ">)" shall be ignored.

Examples. Suppose it is required to input the degenerate interval [1.5,1.5]. Suppose the READ statement is:

```
      INTERVAL X
      READ(*,'(1X,VF18.5)') X
```

Then any of the following input lines results in an internal representation for X that is equal to or contains the interval [1.5,1.5].

```
  1.5
```

or

```
  1.5E0
```

or

```
  (<1.5>)
```

or

```
  (<1.5,1.5>)
```

> Note: It is also possible to use single-number input to describe intervals of width 1 unit in the last place exhibited, and centered on the input value. See the SF edit descriptor below.

C. <u>The Interval VE Edit Descriptor</u>

VE editing is analogous to VF editing, except that it corresponds to the E, rather than F, edit descriptor.

The form and interpretation of the input field shall be the same as for VF editing.

An output field for a VE<w>.<d>[E<e>] edit descriptor shall be of the form ve-output-field, where

5) ve-output-field is (< e-output-field_sub1, e-output-field_sub2 >)

```
6) e-output-field  is the output field for the E<w>.<d>[Ee]
                             format, where <w>, <d>, and <e> are the
                             values specified in the VE<w>.<d>[E<e>]
                             edit descriptor.
```

The value corresponding to e-output-field_sub1 shall be less than or equal to the exact lower bound of the corresponding output list item, and the value corresponding to e-output-field_sub2 shall be greater than or equal to the exact upper bound of the corresponding output list item, regardless of the number of digits in the field and number of digits in the internal representation.

It shall be possible to use the VE edit descriptor to output REAL data. In that case, the value corresponding to e-output-field_sub1 shall be less than or equal to the exact value of the corresponding real output list item, and the value of e-output-field_sub2 shall be greater than or equal to the exact upper bound of the corresponding real output list item, regardless of the number of digits in the field and number of digits in the internal representation.

For both input and output, the symbols "(<" , ">)" and "," that are part of the field shall not be counted as part of the width <w> of the overall VE field. The total width of the field is thus 2<w>+5.

Example: Suppose an interval variable X is defined in a program, suppose the program contained the statement

```
WRITE(*,'(1X,VE12.5E1)') X
```

and suppose the internal representation of X is that of the interval

```
(<1.9921875,2.9921875>)}}
```

Then a valid output produced by the WRITE statement is

```
(< +0.19921E+1, +0.29922E+1>)
```

D. The Interval SE and SF Edit Descriptors

The SE and SF formats are for single number output of INTERVAL's, with an implied error of plus or minus .5 units in the last exhibited digit.

> Note: In a decimal representation, a number with an implied error of plus or minus .5 units in the last exhibited digit corresponds to the set of decimal numbers that are rounded into the exhibited number.

The basic representation of an INTERVAL as a single number is as follows:

a) A single number without a decimal point shall represent an interval whose lower and upper endpoints are identical (a degenerate, i.e. a point interval).

b) A single number with a decimal point shall represent an interval whose endpoints are constructed by subtracting and adding .5 units in the last exhibited decimal digit.

Examples. Here are some intervals represented by single numbers:

Single Number	Interval represented
(<.1>)	[0.05,.15]
(<1>)	[1, 1]
(<.1000>)	[.09995, .10005]
(<1E-1>)	[0.1,0.1]
(<.0>)	[-.05,.05]
(<0.E3>)	[-0.5E3, 0.5E3]

> Note: Degenerate interval decimal constants, such at [0.1,0.1], are not always representable exactly on binary machines. Conversion of such degenerate intervals to internal format is specified by outward rounding, as explained below.

The SE and SF specifiers have the same form as the E and F specifiers, i.e. SF<w>.<d> and SE<w>.<d>[E<e>]. However, the output of an SE or SF specifier shall be of the form

```
        (<single-number-output>)
```

where single-number-output is of the form specified by F<w>.<d> for the SF<w>.<d> specifier, and of the form specified by E<w>.<d>[E<e>] for the SE<w>.<d>[E<e>] specifier, with the following differences:

* Only those leading digits that are equal in the left and right endpoints of the internally represented interval, in the sense

above, shall be printed.

* For zero-width intervals, neither a decimal point nor digits past the decimal point shall be displayed.

* For the SF specifier, the positions defined in the descriptor that correspond to digits that are not displayed shall be filled by blanks, so the displayed digits are justified as though all digits prescribed by the specifier were printed. For the SE specifier, if only s digits are printed, the output will be in the form of an E<w>.<s> specifier, left-justified in the field that it would occupy if s were equal to d.

* If a number is too inaccurate to be represented within a specified SF format, the entire field will be filled with asterisks.

Note: The "SF" format, when used for zero-width intervals, is limited to integers., since all other zero-width intervals will be output as fields of asterisks.

Note: The <w> specifies the width of the numerical field, and not the spaces taken by "(<" and ">)". Thus, the total width of an SE or SF field is <w>+4.

The input for an SE or SF specifier shall be of the form sf-input-field, where:

```
   sf-input-field is f-input-field
                  or (< f-input-field >)

   f-input-field  is  a valid input field for the F<w>.<d>
                        format, where <w> and <d> are the
                        values specified in the SF<w>.<d>
                        or SE<w><d>[E<e>] edit descriptor.
```

Upon input, the internal representation shall depend upon whether the input string contains a decimal point. If the input string contains a decimal point, the number shall be equal to or contain the interval centered upon the number represented by f-input-field, and with width equal to one decimal unit in the least significant digit exhibited. If the input does not contain a decimal point, the internal representation shall be the same as if a VF format specifier is used.

Examples. Suppose the number 1.5 is known to be correct to the last digit represented, to within rounding.
Suppose the READ statement is:

```
      INTERVAL X
      READ(*,'(1X,SF18.5)') X
```

Then any of the following input lines result in an internal representation for X that is equal to or contains the interval [1.45,1.55].

```
  1.5
```

or

```
  1.5E0
```

or

```
  (<1.5>)
```

However, the following inputs merely result in an internal representation for X that is equal to or contains the degenerate interval [1.5,1.5].

```
  15E-1
```

or

```
  (<15E-1>)
```

In a second example, inputs of

```
  0.E1
```

or

```
  (<0.E1>)
```

or

```
  0.0E2
```

or

```
  (<0.0E2>)
```

result in an internal representation for X that contained the interval [-5,5].

> Note: In single number interval I/O, input immediately followed by output can appear to suggest that a decimal digit of accuracy

has been lost, when in fact radix conversion has caused a 1 ulp increase in the width of the interval stored in the machine. For example, an input of 0.100 followed by an immediate print will result in 0.10. Users and implementers need to expect this behavior.

E. The Interval SG and VG Edit Descriptors

(i) The interval SG edit descriptor

The interval SG edit descriptor is for general single-number interval input and output.

For input, the SG edit descriptor is identical to the SF edit descriptor.

The form of the interval SG descriptor is SGw.d or SGw.dEe, where w is the width of the field and d is the maximum number of digits actually displayed. The method of representation of the output field depends both on the magnitude of the datum being edited and on the number of digits that are equal in the lower bound and upper bound. Define the following:

> Round a lower and upper bound to s significant decimal digits. Then if all s digits agree, the s digits are said to CORRESPOND.
>
> Let r be the minimum of d and q, where q is less than or equal to the maximum number of significant digits of the lower and upper bounds of the internal representation that correspond.

If at least one decimal digit of the lower bound and upper bound correspond then the number printed by the SGw.d or SGw.dEe formats shall be the same as that printed by the respective Gw.r and Gw.rEe formats. The printed number is preceded by the string "(<" and is followed by the string ">)".

In determining the number of digits that correspond, the lower and upper bounds of the internal representation may first be converted to a decimal number in such a way that the converted lower bound is less than or equal to the lower bound of the internal representation and the converted upper bound is greater than or equal to the upper bound of the internal representation. In any case, the number actually printed shall have r digits that are equal to the r most significant decimal digits of the lower bound and upper bound of the internal representation.

> Note: Ideally, the number of digits determined to correspond should be the number of digits that actually are equal (in the sense of rounding in the last digit) in the internal representation.

If no digits correspond in the above sense, then the output is the same as with an SEw.0 or SEw.0Ee edit descriptor, left-justified in the field.

Example. If the internal representation equals (<1,100>), then an SG12.5E1 edit descriptor can result in output of the form

```
(<0.E3        >)
```

This output is interpreted as the interval [-500,500].

(ii) <u>The interval VG edit descriptor</u>

The interval VG edit descriptor is identical to the SF edit descriptor for input. For output with VGd.w or VGd.wEe, two decimal numbers are printed, preceded by "(<", separated by ",", and followed by ">)". The formats of the lower bound and upper bound are determined in accordance with the rules for Gd.w or Gd.w.Ee editing respectively, as if the lower bound and upper bound were REAL output. However, the printed lower bound shall be less than or equal to the lower bound of the internal representation, and the printed upper bound shall be greater than or equal to the upper bound of the internal representation.

F. <u>Other Interval I/O</u>

It shall also be possible to input and output INTERVAL data with the E and / or F edit descriptors. In that case, two E or F fields, or one of each, are required for each INTERVAL datum, and the lower and upper bounds of the INTERVAL datum are treated as usual REAL data.

> Note: Warning: In this case the printed interval endpoints may not contain the internal representation.

Intervals may be also be input and output with a "G" edit descriptor. Input of an INTERVAL with the G edit descriptor shall be identical to input with the SF edit descriptor. Output of an interval with a "G" edit descriptor shall be identical to output with an "SG" edit descriptor if one or more digits of the internal representation correspond, and shall be identical to output with the "VG" edit descriptor otherwise.

> Note: The reason the "G" format favors VG when no digits correspond is because the resulting ideal output interval has a smaller width in this case. For example, if the internal representation is [1,2], an SG output gives an interval that contains (<0.E1>) = [-5,5], whereas the output corresponding to a VG specifier can be exact.

INTERVAL's can also be input and output with NAMELIST and list-directed I/O.

Output of intervals in list-directed I/O shall be identical to output with a "G" edit descriptor, with reasonable, processor-dependent values of <w>, <d>, and <e>.

G. An Example

The following program implements a one-dimensional interval Newton method to enclose a solution to x**2 - 4, beginning with starting interval [1,2].

```
PROGRAM INTERVAL_NEWTON_ITERATION_1_D
  IMPLICIT NONE

  REAL(KIND=KIND(0.0D0)) :: XP
  INTERVAL :: X, X_IMAGE
  INTEGER K
  REAL(KIND=KIND(0.0D0)) :: WIDTH_OF_X, OLD_WIDTH

  CALL SIMINI
     WIDTH_OF_X = 4
     X = INTERVAL(1,2)
     DO K = 1,10000
        OLD_WIDTH = WIDTH_OF_X
        WIDTH_OF_X = WID(X)
        XP = MID(X)
        WRITE(*,*) K, INF(X),SUP(X), XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
        X_IMAGE = XP - ( INTERVAL(XP)**2-INTERVAL(4) ) / (2*X)
        IF(WIDTH_OF_X.LT.1D-11) EXIT
        X = X_IMAGE
     END DO
END PROGRAM INTERVAL_NEWTON_ITERATION_1_D
```

The output for this program can be:

Columns 1 through 4:

```
 1   0.9999999999999998   2.0000000000000004   1.5000000000000000
 2   1.9374999999999991   2.3750000000000018   2.1562500000000004
 3   1.9886592741935472   2.0195312500000009   2.0040952620967740
 4   1.9999724292486507   2.0000354537727549   2.0000039415107027
 5   1.9999999999417792   2.0000000000659868   2.0000000000038831
 6   1.9999999999999989   2.0000000000000009   2.0000000000000000
```

Columns 5 and 6:

```
1.0000000000000009   6.2500000000000056E-02
0.4375000000000028   0.4375000000000020
3.0871975806453740E-02   0.1612903225806542
6.3024524104227111E-05   6.6127289936492972E-02
1.2420753314756896E-10   3.1270065174661590E-02
1.9984014443252822E-15   1.2953492022672087E+05
```

        Note: The new iterate is contained in the interior of the old
        iterate between steps 2 and 3. This constitutes a mathematical
        proof that there is a unique solution of x**2-4 = 0 within
        iterate 2, and hence within iterate 3, and all subsequent
        iterates will contain this solution.

For the VE format, replace the line:

```
        WRITE(*,*) K, INF(X),SUP(X), XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
```

by:

```
        WRITE(*,'(1X,I2,1X,VE10.4E1,3(1X,E12.4E2))') &
          K, X, XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
```

The output is then:

```
  1 (<  0.9999E+0, 0.2001E+1 >)  0.1500E+01  0.1000E+01  0.6250E-01
  2 (<  0.1937E+1, 0.2376E+1 >)  0.2156E+01  0.4375E+00  0.4375E+00
  3 (<  0.1987E+1, 0.2020E+1 >)  0.2004E+01  0.3087E-01  0.1612E+00
  4 (<  0.1999E+1, 0.2001E+1 >)  0.2000E+01  0.6302E-04  0.6612E-01
  5 (<  0.1999E+1, 0.2001E+1 >)  0.2000E+01  0.1242E-09  0.3127E-01
  6 (<  0.1999E+1, 0.2001E+1 >)  0.2000E+01  0.1998E-14  0.1295E+05
```

For the SE format, replace the WRITE statement by

```
        WRITE(*,'(1X,I2,1X,SE10.4E1,3(1X,E12.4E2))') &
          K, X, XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
```

The output is then:

```
  1 (<  0.E+1     >)  0.1500E+01  0.1000E+01  0.6250E-01
  2 (<  0.2E+1    >)  0.2156E+01  0.4375E+00  0.4375E+00
  3 (<  0.20E+1   >)  0.2004E+01  0.3087E-01  0.1612E+00
  4 (<  0.2000E+1 >)  0.2000E+01  0.6302E-04  0.6612E-01
  5 (<  0.2000E+1 >)  0.2000E+01  0.6302E-04  0.6612E-01
  6 (<  0.2000E+1 >)  0.2000E+01  0.6302E-04  0.6612E-01
```

For the SF format, replace the WRITE statement by

```
        WRITE(*,'(1X,I2,1X,SF18.15,3(1X,E12.4E2))') &
          K, X, XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
```

The output is then:

```
 1 (<******************>)  0.1500E+01  0.1000E+01  0.6250E-01
 2 (< 2.               >)  0.2156E+01  0.4375E+00  0.4375E+00
 3 (< 2.0              >)  0.2004E+01  0.3087E-01  0.1612E+00
 4 (< 2.0000           >)  0.2000E+01  0.6302E-04  0.6612E-01
 5 (< 2.000000000      >)  0.2000E+01  0.1242E-09  0.3127E-01
 6 (< 2.00000000000000 >)  0.2000E+01  0.1998E-14  0.1295E+05
```

The number of digits printed in this case is the number of digits known to be correct, assuming the actual number, displayed as an infinite decimal sequence, has been rounded into the displayed digits by rounding to nearest.

For the SE format, replace the WRITE statement by

```
        WRITE(*,'(1X,I2,1X,SG18.15,3(1X,E12.4E2))') &
          K, X, XP, WIDTH_OF_X, WIDTH_OF_X/OLD_WIDTH**2
```

and the output can be:

```
 1 (< 0.E+1            >)  0.1500E+01  0.1000E+01  0.6250E-01
 2 (< 2.               >)  0.2156E+01  0.4375E+00  0.4375E+00
 3 (< 2.0              >)  0.2004E+01  0.3087E-01  0.1612E+00
 4 (< 2.0000           >)  0.2000E+01  0.6302E-04  0.6612E-01
 5 (< 2.000000000      >)  0.2000E+01  0.1242E-09  0.3127E-01
 6 (< 2.00000000000000 >)  0.2000E+01  0.1998E-14  0.1295E+05
```

V. ADDITIONAL NOTES (not part of proposal proper)

A. On Optimization of Interval Expressions

Optimizing compilers generally attempt to reduce the total number of point operations. However, since interval arithmetic is only subdistributive, there are additional issues in optimizing interval expressions. For example,

```
                          (<0,1>)**2 - (<0,1>)
evaluates to
                     (<0,1>) - (<0,1>) = (<-1,1>),
while
                     (<0,1>) * ((<0,1>)-(<1,1>))
evaluates to
                     (<0,1>) * (<-1,0>) = (<-1,0>).
```

However, both (<-1,1>) and (<-1,0>) are bounds on the range of x**2-x over (<0,1>). We want to transform x**2-x to x*(x-1) in this case, or else compute the expression BOTH ways and take the intersection, to obtain the smallest possible enclosure to the range of x**2 - x over (<0,1>). Such rewriting should be possible with modifications of existing optimizing compiler technology.

B. Can Interval Arithmetic be Implemented Effectively in a Module?

The following items appear central to this question:

* INTERVAL constants are not implementable in a module.

* The I/O edit descriptors cannot be defined in a module.

* The precedence of the new infix operators cannot be defined in a module.

* There is an efficiency issue.

Although edit descriptors in Fortran 95 are fixed, INTERVAL I/O can be defined in a module through subroutines. Furthermore, derived-type I/O, discussed for Fortran 2000, would ameliorate the situation with I/O descriptors, should derived-type I/O materialize.

There is no way other than intrinsic language support to allow INTERVAL constants in program statements. INTERVAL constants can be supported in module subroutines for input and output.

However, a crucial property of such constants cannot be easily implemented in a module. Namely, the proposal stipulates that conversions to and from internal representations shall contain the original results, regardless of the number of digits present in the internal representation or the number of digits in the decimal constant or format item.

Regarding efficiency: User-supplied modules for interval arithmetic contain subroutines for each of the four arithmetic operations and for the other infix operators. At least one existing compiler translates each elementary operation to a subroutine call to the corresponding subroutine. The resulting machine code executes 20 or more times more slowly than point arithmetic, although factors of five or even two are often practical. It can be argued that an optimizing compiler will in-line short subroutines. The concept of an INTRINSIC MODULE, that is, a module supplied by the manufacturer and bundled with the compiler, has been put forward for Fortran 2000. Presumably, with an intrinsic module, there would be essentially no difference between implementation of interval arithmetic intrinsically in the language and as a module, except that access to the interval arithmetic would be enabled through a "USE" statement. However, we are unaware of in-lining Fortran compilers, and intrinsic modules have not yet materialized as part of the language.

There is a natural operator precedence for the INTERVAL operators that differs from that for user-defined operators in Fortran 90 (i.e. user-defined binary operations always have the lowest precedence). This order, followed in aACRITH-XSC, is:

```
Numeric                   **
Numeric                  *, /, .IS.
Numeric              unary + or -
Numeric             binary + or -, .CH.
Character                 //
Relational .EQ.,.NE.,.LT.,.LE.,.GT.,.GE.,.SB.,.SP.,.DJ.,.IN.                 .
                 ==, /=, <, <=, >, >=
Logical                 .NOT.
Logical                 .AND.
Logical                 .OR.
Logical             .EQV. or .NEQV.
```

At present, such an ordering can be defined only intrinsically in the language, not through a module.