12. Input and output (I/O) of intervals

12.1. Overview. This clause of the standard specifies conversion from a text string that holds an interval literal to an interval internal to a program (input), and the reverse (output). The methods by which strings are read from, or written to, a character stream are language- or implementation-defined, as are variations in some locales (such as specific character case matching).

Containment is preserved on input and output so that, when a program computes an enclosure of some quantity given an enclosure of the data, it can ensure this holds all the way from text data to text results.

In addition to the above I/O, which may incur rounding errors on output and/or input, each interval type $\mathbb{T}$ has an *exact text representation*, via operations that convert any internal $\mathbb{T}$-interval $\boldsymbol{x}$ to a string $\boldsymbol{s}$, and back again to recover $\boldsymbol{x}$ exactly. For each type, the implementation is required to document the mathematical relation between $\boldsymbol{s}$ and the Level 1 interval value of $\boldsymbol{x}$.

12.2. Input. Input is provided for each supported bare or decorated interval type $\mathbb{T}$ by the $\mathbb{T}$-version of `textToInterval`($\boldsymbol{s}$), where $\boldsymbol{s}$ is a string, as specified in §11.11.9. It accepts an arbitrary interval literal $\boldsymbol{s}$ and returns a $\mathbb{T}$-interval enclosing the Level 1 value of $\boldsymbol{s}$.

For 754-conforming types $\mathbb{T}$ the required tightness is specified in §11.11.9. For other types the tightness is implementation-defined.

[*Note. This provides the basis for free-format input of interval literals from a text stream, as might be provided by overloading the >> operator in C++.*]

12.3. Output. An implementation shall provide an operation

`intervalToText`($\boldsymbol{X}$, *cs*)

where *cs* is optional. $\boldsymbol{X}$ is a bare or decorated interval datum of any supported interval type $\mathbb{T}$, and *cs* is a string, the conversion specifier. The operation converts $\boldsymbol{X}$ to a valid interval literal string $\boldsymbol{s}$, see§11.11.1, which shall be related to $\boldsymbol{X}$ as follows, where $\boldsymbol{Y}$ is the Level 1 value of $\boldsymbol{s}$.

(i) Let $\mathbb{T}$ be a bare type. Then $\boldsymbol{Y}$ shall contain $\boldsymbol{X}$, and shall be empty if $\boldsymbol{X}$ is empty.

(ii) Let $\mathbb{T}$ be a decorated type. If $\boldsymbol{X}$ is NaI then $\boldsymbol{Y}$ shall be NaI. Otherwise, write $\boldsymbol{X} = \boldsymbol{x}_{dx}$, $\boldsymbol{Y} = \boldsymbol{y}_{dy}$. Then
- $\boldsymbol{y}$ shall contain $\boldsymbol{x}$, and shall be empty if $\boldsymbol{x}$ is empty.
- dy shall equal dx, except in the case that $dx =$ `com` and overflow occurred, that is, $\boldsymbol{x}$ is bounded and $\boldsymbol{y}$ is unbounded. Then dy shall equal `dac`.

[*Note.* $\boldsymbol{Y}$ *being a Level 1 value is significant. E.g., for a bare type* $\mathbb{T}$*, it is not allowed to convert* $\boldsymbol{X} = \emptyset$ *to the string* `garbage`*, even though converting* `garbage` *back to a bare interval at Level 2 by* $\mathbb{T}$`-textToInterval` *gives* $\emptyset$*, because* `garbage` *has no Level 1 value as a bare interval literal.*]

The tightness of enclosure of $\boldsymbol{X}$ by the value $\boldsymbol{Y}$ of $\boldsymbol{s}$ is language- or implementation-defined.

If present, *cs* controls the layout of the string $\boldsymbol{s}$ and its syntactic parts, see§11.11.1, for instance l, u, m and r where relevant. The implementation shall document the possible values of *cs* and their effect. The standard does not specify *cs* but among the user-controllable elements should be the following.

(i) It should be possible to specify the preferred overall field width (the length of $\boldsymbol{s}$), and whether output is in inf-sup or uncertain form.

(ii) It should be possible to specify how Empty, Entire and NaI are output, e.g., whether lower or upper case, and whether Entire becomes `[Entire]` or `[-Inf, Inf]`.

(iii) For l, u, m and r, it should be possible to specify the field width, and the number of places after the point or the number of significant figures. There should be a choice of radix, at least between decimal and hexadecimal.

(iv) *cs* should provide an option to output the bounds of an interval without punctuation, e.g. `1.234  2.345` instead of `[1.234, 2.345]`. For instance this might be a convenient way to write intervals to a file for use by another application.

If *cs* is absent, output should be in a general-purpose layout (analogous, e.g., to the `%g` specifier of `fprintf` in C). There should be a value of *cs* that selects this layout explicitly.

[*Note. This provides the basis for free-format output of intervals to a text stream, as might be provided by overloading the << operator in C++.*]

DRAFT 7.4

If $\mathbb{T}$ is a 754-conforming bare type, there shall be a value of *cs* that produces behavior identical with that of `intervalToExact`, below. That is, the output is an interval literal that, when read back by $\mathbb{T}$-`textToInterval`, recovers the original datum exactly.

12.4. Exact text representation. For any supported bare interval type $\mathbb{T}$ an implementation shall provide operations `intervalToExact` and `exactToInterval`. Their purpose is to provide a portable exact representation of every bare interval datum in text form.

These operations shall obey the **recovery requirement**:

> For any $\mathbb{T}$-datum $\boldsymbol{x}$, the value $\boldsymbol{s} = \mathbb{T}$-`intervalToExact`$(\boldsymbol{x})$ is a string,
> such that $\boldsymbol{y} = \mathbb{T}$-`exactToInterval`$(\boldsymbol{s})$ is defined and equals $\boldsymbol{x}$.

[*Note. From §11.3, this is equality as datums: $\boldsymbol{x}$ and $\boldsymbol{y}$ have the same Level 1 value and the same type. They may differ at Level 3, e.g., a zero endpoint might be stored as* -0 *in one and* $+0$ *in the other.*]

Suppose $\mathbb{T}$ is a 754-conforming type. Then the string $\boldsymbol{s}$ shall be an interval literal, and the operation `exactToInterval` shall be the same as `textToInterval`. If $\boldsymbol{x}$ is nonempty, $\boldsymbol{s}$ shall be of inf-sup form $[l, u]$ with l and u represented exactly. If $\mathbb{T}$ is a binary type, l and u if finite may be represented as decimal numbers if convenient (e.g., if they are small integers) and otherwise shall be represented in the hexadecimal-significand form of 754§5.12.3. If $\mathbb{T}$ is a decimal type, they shall be represented as decimal numbers.

If $\mathbb{T}$ is not 754-conforming, there are no restrictions on the form of the string $\boldsymbol{s}$ apart from the above recovery requirement. However, the representation should aim to display the values of the parameters that define the underlying mathematical model, in a human-readable way.

The algorithm by which `intervalToExact` converts $\boldsymbol{x}$ to $\boldsymbol{s}$ is regarded as part of the definition of the type and shall be documented by the implementation.

[*Example. Writing a* `binary64` *floating point datum exactly in hexadecimal-significand form passes the "readability" test since it displays the parameters sign, exponent and significand. Dumping its 64 bits as 16 hex characters does not.*]

Since `exactToInterval` creates an interval from non-interval data, it is a constructor similar to `textToInterval`, and (see §11.11.9), shall return Empty and signal a language- or implementation-defined exception when its input is invalid.

12.4.1. *Exact representations of comparable types.* The exact text representation of a bare interval of any type should also be a valid exact representation in any wider (in the sense of §11.5.1) type, which when converted back produces the mathematically same interval.

That is, let type $\mathbb{T}'$ be wider than type $\mathbb{T}$. Let $\boldsymbol{x}$ be a $\mathbb{T}$-interval and let

$$\boldsymbol{s} = \mathbb{T}\text{-}\texttt{intervalToExact}(\boldsymbol{x}).$$

Then

$$\boldsymbol{x}' = \mathbb{T}'\text{-}\texttt{exactToInterval}(\boldsymbol{s})$$

should be defined and equal to $\mathbb{T}'$-`convertType`(x).

[*Note. If $\mathbb{T}$ and $\mathbb{T}'$ are 754-conforming types, this property holds automatically, because of the properties of* `textToInterval` *and the fact that $\boldsymbol{s}$ is an interval literal.*]

DRAFT 7.4