INTERNATIONAL STANDARD

ISO/IEC 10967-1

First edition
1994-12-15

Information technology — Language independent arithmetic —

Part 1:
Integer and floating point arithmetic

Technologies de l'information — Arithmétique indépendante de langage —
Partie 1: Arithmétique de nombres entiers et en virgule flottante

Reference number
ISO/IEC 10967-1:1994(E)

Contents

Annexes

Foreword

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

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

International Standard ISO/IEC 10967-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

ISO/IEC 10967 consists of the following parts, under the general title *Information technology — Language independent arithmetic*:

— *Part 1: Integer and floating point arithmetic*

— *Part 2: Mathematical procedures*

— *Part 3: Complex arithmetic and procedures*

Additional parts will specify other arithmetic types or operations.

Annexes A to J of this part of ISO/IEC 10967 are for information only.

Introduction

The aims

Programmers writing programs that perform a significant amount of numeric processing have often not been certain how a program will perform when run under a given language processor. Programming language standards have traditionally been somewhat weak in the area of numeric processing, seldom providing an adequate specification of the properties of arithmetic data types, particularly floating point numbers. Often they do not even require much in the way of documentation of the actual arithmetic data types by a conforming language processor.

It is the intent of this part of ISO/IEC 10967 to help to redress these shortcomings, by setting out precise definitions of integer and floating point data types, and requirements for documentation. This is done in a way that makes as few presumptions as possible about the underlying machine architecture.

It is not claimed that this part of ISO/IEC 10967 will ensure complete certainty of arithmetic behavior in all circumstances; the complexity of numeric software and the difficulties of analysing and proving algorithms are too great for that to be attempted. Rather, the requirements set forth here will provide a firmer basis than hitherto for attempting such analysis.

Hence the first aim of this part of ISO/IEC 10967 is to enhance the predictability and reliability of the behavior of programs performing numeric processing.

The second aim, which helps to support the first, is to help programming language standards to express the semantics of arithmetic data types. These semantics need to be precise enough for numerical analysis, but not so restrictive as to prevent efficient implementation of the language on a wide range of platforms.

The third aim is to help enhance the portability of programs that perform numeric processing across a range of different platforms. Improved predictability of behavior will aid programmers designing code intended to run on multiple platforms, and will help in predicting what will happen when such a program is moved from one conforming language processor to another.

Note that this part of ISO/IEC 10967 does not attempt to ensure bit-for-bit identical results when programs are transferred between language processors, or translated from one language into another. Programming languages and platforms are too diverse to make that a sensible goal. However, experience shows that diverse numeric environments can yield comparable results under most circumstances, and that with careful program design significant portability is actually achievable.

The content

This part of ISO/IEC 10967 defines the fundamental properties of integer and floating point numbers. These properties are presented in terms of a parameterized model. The parameters allow enough variation in the model so that most platforms are covered, but when a particular set of parameter values is selected, and all required documentation is supplied, the resulting information should be precise enough to permit careful numerical analysis.

The requirements of this part of ISO/IEC 10967 cover three areas. First, the programmer must be given runtime access to the parameters and functions that describe the arithmetic properties of the platform. Second, the executing program must be notified when proper results cannot be returned (e.g., when a computed result is out of range or undefined). Third, the numeric properties of conforming platforms must be publicly documented.

This part of ISO/IEC 10967 focuses on the classical integer and floating point data types. Later parts will consider common mathematical procedures (part 2), complex numbers (part 3), and possibly additional arithmetic types such as fixed point.

Relationship to hardware

ISO/IEC 10967 is not a hardware architecture standard. It makes no sense to talk about an "LIA machine." Future platforms are expected either to duplicate existing architectures, or to satisfy high quality architecture standards such as IEC 559 (also known as IEEE 754). The floating point requirements of this part of ISO/IEC 10967 are compatible with (and enhance) IEC 559.

This part of ISO/IEC 10967 provides a bridge between the abstract view provided by a programming language standard and the precise details of the actual arithmetic implementation.

The benefits

Adoption and proper use of this part of ISO/IEC 10967 can lead to the following benefits.

Language standards will be able to define their arithmetic semantics more precisely without preventing the efficient implementation of their language on a wide range of machine architectures.

Programmers of numeric software will be able to assess the portability of their programs in advance. Programmers will be able to trade off program design requirements for portability in the resulting program.

Programs will be able to determine (at run time) the crucial numeric properties of the implementation. They will be able to reject unsuitable implementations, and (possibly) to correctly characterize the accuracy of their own results. Programs will be able to extract apparently implementation dependent data (such as the exponent of a floating point number) in an implementation independent way. Programs will be able to detect (and possibly correct for) exceptions in arithmetic processing.

End users will find it easier to determine whether a (properly documented) application program is likely to execute satisfactorily on their platform. This can be done by comparing the documented requirements of the program against the documented properties of the platform.

Finally, end users of numeric application packages will be able to rely on the correct execution of those packages. That is, for correctly programmed algorithms, the results are reliable if and only if there is no notification.

Annex A is intended to be read in parallel with the standard.

Information technology – Language independent arithmetic – Part 1: Integer and floating point arithmetic

1 Scope

This part of ISO/IEC 10967 defines the properties of integer and floating point data types on computer systems to ensure that the processing of arithmetic data can be undertaken in a reliable and predictable manner. Emphasis is placed on documenting the existing variation between systems, not on the elimination of such variation. The requirements of this part of ISO/IEC 10967 shall be in addition to those that may be specified in other standards, such as those for programming languages (See clause 7).

It is not the purpose of this part of ISO/IEC 10967 to ensure that an arbitrary numerical function can be so encoded as to produce acceptable results on all conforming systems. Rather, the goal is to ensure that the properties of arithmetic on a conforming system are made available to the programmer.

Therefore, it is not reasonable to demand that a substantive piece of software run on every implementation that can claim conformity to this part of ISO/IEC 10967.

An implementor may choose any combination of hardware and software support to meet the specifications of this part of ISO/IEC 10967. It is the arithmetic environment, as seen by the user, that does or does not conform to the specifications.

The term *implementation* (of this part of ISO/IEC 10967) denotes the total arithmetic environment, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and all pertinent documentation.

1.1 Specifications included in this part of ISO/IEC 10967

This part of ISO/IEC 10967 defines integer and floating point data types. Definitions are included for bounded, unbounded, and modulo integer types, as well as both normalized and denormalized floating point types.

The specification for an arithmetic type includes

a) The set of computable values.

b) The set of computational operations provided, including

   1) primitive operations (addition, subtraction, etc.) with operands of the same type,

2) comparison operations on two operands of the same type,
3) conversion operations from any arithmetic type to any other arithmetic type, and
4) operations that access properties of individual values.

c) Program-visible parameters that characterize the values and operations.

d) Procedures for reporting arithmetic exceptions.

NOTE – A.1.3 describes planned future work in this area.

1.2 Specifications not within the scope of this part of ISO/IEC 10967

This part of ISO/IEC 10967 provides no specifications for

a) Arithmetic and comparison operations whose operands are of more than one data type. This part of ISO/IEC 10967 neither requires nor excludes the presence of such "mixed operand" operations.

b) A general unnormalized floating point data type, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.

c) An interval data type, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.

d) A fixed point data type, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.

e) A rational data type, or the operations on such data. This part of ISO/IEC 10967 neither requires nor excludes such data or operations.

f) The properties of arithmetic data types that are not related to the numerical process, such as the representation of values on physical media.

g) Floating point values that represent infinity or non-numeric results. However, specifications for such values are given in IEC 559.

h) The properties of integer and floating point data types that properly belong in language standards. Examples include

1) The syntax of literals and expressions.
2) The precedence of operators.
3) The rules of assignment and parameter passing.
4) The presence or absence of automatic type coercions.
5) The consequences of applying an operation to values of improper type, or to uninitialized data.

NOTE – See clause 7 and annex E for a discussion of language standards and language bindings.

The internal representation of values is beyond the scope of this part of ISO/IEC 10967. Internal representations need not be unique, nor is there a requirement for identifiable fields (for sign, exponent, and so on). The value of the exponent bias, if any, is not specified.

2 Conformity

It is expected that the provisions of this part of ISO/IEC 10967 will be incorporated by reference and further defined in other International Standards; specifically in language standards and in language binding standards. Binding standards specify the correspondence between the abstract data types and operations of this part of ISO/IEC 10967 and the concrete language syntax of the language standard. A language standard that explicitly provides such binding information can serve as a binding standard.

When a binding standard for a language exists, an implementation shall be said to conform to this part of ISO/IEC 10967 if and only if it conforms to the binding standard. In particular, in the case of conflict between a binding standard and this part of ISO/IEC 10967, the specifications of the binding standard shall take precedence.

When no binding standard for a language exists, an implementation conforms to this part of ISO/IEC 10967 if and only if it provides one or more data types that together satisfy all the requirements of clauses 5 through 8. Conformity is relative to that designated set of data types.

NOTES

1 Language bindings are essential. Clause 8 requires an implementation to supply a binding if no binding standard exists. See clause A.7 for recommendations on the proper content of a binding standard. See annex F for an example of a conformity statement, and annex E for suggested language bindings.

2 A complete binding for this part of ISO/IEC 10967 will include a binding for IEC 559 as well. See 5.2.9 and annex C.

An implementation is free to provide arithmetic types that do not conform to this part of ISO/IEC 10967 or that are beyond the scope of this part of ISO/IEC 10967. The implementation shall not claim conformity for such types.

An implementation is permitted to have modes of operation that do not conform to this part of ISO/IEC 10967. However, a conforming implementation shall specify how to select the modes of operation that ensure conformity.

3 Normative reference

The following standard contains provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10967. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 10967 are encouraged to investigate the possibility of applying the most recent edition of the standard indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

IEC 559:1989, *Binary floating-point arithmetic for microprocessor systems.*

4 Symbols and definitions

4.1 Symbols

In this part of ISO/IEC 10967, $\mathcal{Z}$ denotes the set of mathematical integers, $\mathcal{R}$ denotes the set of real numbers, and $\mathcal{C}$ denotes the set of complex numbers. Note that $\mathcal{Z} \subset \mathcal{R} \subset \mathcal{C}$.

All prefix and infix operators have their conventional (exact) mathematical meaning. The conventional notation for set definition and manipulation is also used. In particular this part of ISO/IEC 10967 uses

$\Rightarrow$ and $\Longleftrightarrow$ for logical implication and equivalence
$+$, $-$, $*$, $/$, x^y, $log_x y$, $\sqrt{x}$, $|x|$, $\lfloor x \rfloor$, and $tr(x)$ on reals
$<$, $\leq$, $=$, $\neq$, $\geq$, and $>$ on reals
$\times$, $\cup$, $\in$, $\subset$, and $=$ on sets of integers and reals
max and min on non-empty sets of integers and reals
$\rightarrow$ for a mapping between sets

This part of ISO/IEC 10967 uses $*$ for multiplication, and $\times$ for the Cartesian product of sets.

For $x \in \mathcal{R}$, the notation $\lfloor x \rfloor$ designates the largest integer not greater than x:

$$\lfloor x \rfloor \in \mathcal{Z} \quad \text{and} \quad x - 1 < \lfloor x \rfloor \leq x$$

and $tr(x)$ designates the integer part of x (truncated toward 0):

$$\begin{aligned} tr(x) &= \lfloor x \rfloor && \text{if } x \geq 0 \\ &= -\lfloor -x \rfloor && \text{if } x < 0 \end{aligned}$$

The type *Boolean* consists of the two values **true** and **false**. Predicates (like $<$ and $=$) produce values of type Boolean.

4.2 Definitions

For the purposes of this part of ISO/IEC 10967, the following definitions apply:

arithmetic data type: A data type whose values are members of $\mathcal{Z}$, $\mathcal{R}$, or $\mathcal{C}$.

NOTE 1 – This part of ISO/IEC 10967 specifies requirements for integer and floating point data types. Complex numbers are not covered here, but will be included in a subsequent part of ISO/IEC 10967 [15].

axiom: A general rule satisfied by an operation and all values of the data type to which the operation belongs. As used in the specifications of operations, axioms are requirements.

continuation value: A computational value used as the result of an arithmetic operation when an exception occurs. Continuation values are intended to be used in subsequent arithmetic processing. (Contrast with *exceptional value*. See 6.1.2.)

NOTE 2 – The infinities and NaNs produced by an IEC 559 system are examples of continuation values.

data type: A set of values and a set of operations that manipulate those values.

denormalization loss: A larger than normal rounding error caused by the fact that denormalized values have less than full precision. (See 5.2.5 for a full definition.)

denormalized: Those values of a floating point type F that provide less than the full precision allowed by that type. (See F_D in 5.2 for a full definition.)

error: (1) The difference between a computed value and the correct value. (Used in phrases like "rounding error" or "error bound.")

(2) A synonym for *exception* in phrases like "error message" or "error output." Error and exception are not synonyms in any other context.

exception: The inability of an operation to return a suitable numeric result. This might arise because no such result exists mathematically, or because the mathematical result cannot be represented with sufficient accuracy.

exceptional value: A non-numeric value produced by an arithmetic operation to indicate the occurrence of an exception. Exceptional values are not used in subsequent arithmetic processing. (See clause 5.)

NOTES

3 Exceptional values are used as part of the defining formalism only. With respect to this part of ISO/IEC 10967, they do not represent values of any of the data types described. There is no requirement that they be represented or stored in the computing system.

4 Exceptional values are not to be confused with the NaNs and infinities defined in IEC 559. Contrast this definition with that of *continuation value* above.

exponent bias: A number added to the exponent of a floating point number, usually to transform the exponent to an unsigned integer.

helper function: A function used solely to aid in the expression of a requirement. Helper functions are not visible to the programmer, and are not required to be part of an implementation. However, some implementation defined helper functions are required to be documented.

implementation (of this part of ISO/IEC 10967)**:** The total arithmetic environment presented to a programmer, including hardware, language processors, exception handling facilities, subroutine libraries, other software, and all pertinent documentation.

normalized: Those values of a floating point type F that provide the full precision allowed by that type. (See F_N in 5.2 for a full definition.)

notification: The process by which a program (or that program's user) is informed that an arithmetic exception has occurred. For example, dividing 2 by 0 results in a notification. (See clause 6 for details.)

operation: A function directly available to the user, as opposed to helper functions or theoretical mathematical functions.

precision: The number of digits in the fraction of a floating point number. (See 5.2.)

rounding: The act of computing a representable final result for an operation that is close to the exact (but unrepresentable) result for that operation. Note that a suitable representable result may not exist (see 5.2.6). (See also A.5.2.5 for some examples.)

rounding function: Any function $rnd : \mathcal{R} \rightarrow X$ (where X is a discrete subset of $\mathcal{R}$) that maps each element of X to itself, and is monotonic non-decreasing. Formally, if x and y are in $\mathcal{R}$,

$$x \in X \Rightarrow rnd(x) = x$$
$$x < y \Rightarrow rnd(x) \leq rnd(y)$$

Note that if $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects one of those adjacent values.

round to nearest: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one nearest u. If the adjacent values are equidistant from u, either may be chosen.

round toward minus infinity: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one less than u.

round toward zero: The property of a rounding function rnd that when $u \in \mathcal{R}$ is between two adjacent values in X, $rnd(u)$ selects the one nearest 0.

shall: A verbal form used to indicate requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted. (Quoted from [2].)

should: A verbal form used to indicate that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that (in the negative form) a certain possibility is deprecated but not prohibited. (Quoted from [2].)

signature (of a function or operation)**:** A summary of information about an operation or function. A signature includes the operation name, the minimum set of inputs to the operation, and the maximum set of outputs from the operation (including exceptional values if any). The signature

$$add_I : I \times I \to I \cup \{\mathbf{integer\_overflow}\}$$

states that the operation named add_I shall accept any pair of I values as input, and (when given such input) shall return either a single I value as its output or the exceptional value **integer_overflow**.

A signature for an operation or function does not forbid the operation from accepting a wider range of inputs, nor does it guarantee that every value in the output range will actually be returned for some input. An operation given inputs outside the stipulated input range may produce results outside the stipulated output range.

5 The arithmetic types

A type consists of a set of values and a set of operations that manipulate these values. For any particular type, the set of values is characterized by a small number of parameters. An exact definition of the value set will be given in terms of these parameters.

Given the type's value set (V), the type's operations will be specified as a collection of mathematical functions on V. These functions typically return values in V, but they may instead return certain "exceptional" values that are not in any arithmetic type. The exceptional values are **integer_overflow**, **floating_overflow**, **underflow**, and **undefined**.

NOTES

1 Exceptional values are used as part of the defining formalism only. With respect to this part of ISO/IEC 10967, they do not represent values of any of the data types described. There is no requirement that they be represented or stored in the computing system. They are not used in subsequent arithmetic operations.

2 The values *Not-a-Number* and *infinity* introduced in IEC 559 (also known as IEEE 754) are not considered exceptional values for the purposes of this part of ISO/IEC 10967. They are "continuation values" as defined in 6.1.2.

Whenever an arithmetic operation (as defined in this clause) returns an exceptional value, notification of this shall occur as described in clause 6.

Each operation has a signature which describes its inputs and outputs (including exceptional values). Each operation is further defined by one or more axioms.

An implementation of a conforming integer or floating point type shall include all the values defined for that type in this part of ISO/IEC 10967. Additional numeric values shall not be included in such a type. However, an implemented type is permitted to include additional non-numeric values (for example, continuation values representing exceptions).

NOTE 3 – This part of ISO/IEC 10967 does not define the behavior of operations when applied to such additional non-numeric values. Other standards (such as IEC 559) do define such behavior.

An implementation of a conforming integer or floating point type shall include all the operations defined for that type in this part of ISO/IEC 10967. Additional operations are explicitly permitted.

The type Boolean is used to specify parameter values and the results of comparison operations. An implementation is not required to provide a Boolean type, nor is it required to provide operations on boolean values. However, an implementation shall provide a means of distinguishing **true** from **false** as parameter values and as results of operations.

NOTE 4 – This part of ISO/IEC 10967 requires an implementation to provide "means" or "methods" to access values, operations, or other facilities. Ideally, these methods are provided by a language or binding standard, and the implementation merely cites these standards. Only if a binding standard does not exist, must an individual implementation supply this information on its own. See clause A.7.

5.1 Integer types

An integer type I shall be a subset of $\mathcal{Z}$, characterized by four parameters:

$bounded \in Boolean$	(whether the set I is finite)
$modulo \in Boolean$	(whether out-of-bounds results "wrap")
$minint \in I$	(the smallest integer in I)
$maxint \in I$	(the largest integer in I)

If *bounded* is **false**, the set I satisfies

$$I = \mathcal{Z}$$

In this case, *modulo* shall be **false**, and the values of *minint* and *maxint* are not meaningful.

If *bounded* is **true**, the set I satisfies

$$I = \{x \in \mathcal{Z} \mid minint \le x \le maxint\}$$

and *minint* and *maxint* shall satisfy

$$maxint > 0$$

and one of:
$$minint = 0$$
$$minint = -(maxint)$$
$$minint = -(maxint + 1)$$

An integer type with $minint < 0$ is called *signed*. An integer type with $minint = 0$ is called *unsigned*. An integer type in which *bounded* is **false** is *signed*.