

Implementation of
ISO/IEC 10967-1:1994

Information technology — Language independent arithmetic —

Part 1: Integer and floating point
arithmetic

ICS 35.060.00

NO COPYING WITHOUT BSI PERMISSION EXCEPT AS PERMITTED BY COPYRIGHT LAW



Document Sharing Hub

Committees responsible for this British Standard

The preparation of this British Standard was entrusted to Technical Committee IST/5, Programming languages, their environments and system software interfaces, upon which the following bodies were represented:

- British Computer Society
- Department of Trade and Industry (National Engineering Laboratory)
- Electricity Association
- Hoskyns Group plc
- IBM United Kingdom Ltd.
- International Computers Ltd.
- Joint Information Systems Committee
- National Computer Users Forum
- National Computing Centre Ltd.
- Rutherford Appleton Laboratory
- SEMA Group Systems Ltd.

This British Standard, having been prepared under the direction of the Information Systems Technology Assembly, was published under the authority of the Standards Board and comes into effect on 15 June 1995

© BSI 03-2000

The following BSI references relate to the work on this standard:
Committee reference IST/5
Draft for comment 93/643027 DC

ISBN 0 580 24404 0

Amendments issued since publication

Amd. No.	Date	Comments

Contents

	Page
Committees responsible	Inside front cover
National foreword	ii
Foreword	v
Text of ISO/IEC 10967-1	1

National foreword

This British Standard reproduces verbatim ISO/IEC 10967-1:1994 and implements it as the UK national standard.

This British Standard is published under the direction of the Information Systems Technology Assembly whose Technical Committee IST/5 has the responsibility to:

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

NOTE International and European Standards, as well as overseas standards, are available from Customer Services, BSI, 389 Chiswick High Road, London W4 4AL.

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

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

Summary of pages

This document comprises a front cover, an inside front cover, pages i and ii, the ISO/IEC title page, pages ii to vi, pages 1 to 71 and a back cover.

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

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

	Page
Foreword	v
Introduction	1
1 Scope	2
1.1 Specifications included in this part of ISO/IEC 10967	2
1.2 Specifications not within the scope of this part of ISO/IEC 10967	3
2 Conformity	3
3 Normative reference	4
4 Symbols and definitions	4
4.1 Symbols	4
4.2 Definitions	4
5 The arithmetic types	6
5.1 Integer types	7
5.1.1 Operations	7
5.1.2 Modulo integers versus overflow	8
5.1.3 Axioms	8
5.2 Floating point types	9
5.2.1 Range and granularity constants	10
5.2.2 Operations	10
5.2.3 Approximate operations	11
5.2.4 Approximate addition	11
5.2.5 Rounding	11
5.2.6 Result function	12
5.2.7 Axioms	12
5.2.8 Rounding constants	13
5.2.9 Conformity to IEC 559	14
5.3 Conversion operations	14
6 Notification	15
6.1 Notification alternatives	15
6.1.1 Language defined notification	15
6.1.2 Recording of indicators	16
6.1.3 Termination with message	17
6.2 Delays in notification	17
6.3 User selection of alternative for notification	17
7 Relationship with language standards	17
8 Documentation requirements	18
Annex A (informative) Rationale	20
A.1 Scope	20
A.1.1 Specifications included in this part of ISO/IEC 10967	20
A.1.2 Specifications not within the scope of this part of ISO/IEC 10967	20
A.1.3 Proposed follow-ons to this part of ISO/IEC 10967	20
A.2 Conformity	21
A.2.1 Validation	21
A.3 Normative references	22
A.4 Symbols and definitions	22
A.4.1 Symbols	22
A.4.2 Definitions	22
A.5 The arithmetic types	23
A.5.1 Integer types	23
A.5.1.1 Operations	24
A.5.1.2 Modulo integers versus overflow	24

	Page
A.5.1.3 Axioms	25
A.5.2 Floating point types	25
A.5.2.1 Range and granularity constants	28
A.5.2.2 Operations	29
A.5.2.3 Approximate operations	30
A.5.2.4 Approximate addition	30
A.5.2.5 Rounding	31
A.5.2.6 Result function	31
A.5.2.7 Axioms	32
A.5.2.8 Rounding constants	32
A.5.2.9 Conformity to IEC 559	32
A.5.2.10 Relations among floating point types	32
A.5.2.11 Levels of predictability	33
A.5.2.12 Identities	33
A.5.2.13 Precision, accuracy, and error	35
A.5.2.14 Extra precision	37
A.5.3 Conversion operations	38
A.6 Notification	38
A.6.1 Notification alternatives	38
A.6.1.1 Language defined notification	38
A.6.1.2 Recording of indicators	38
A.6.1.3 Termination with message	39
A.6.2 Delays in notification	39
A.6.3 User selection of alternative for notification	40
A.7 Relationship with language standards	40
A.8 Documentation requirements	41
Annex B (informative) Partial conformity	41
Annex C (informative) IEC 559 bindings	42
C.1 Summary	42
C.2 Notification	43
C.3 Rounding	43
Annex D (informative) Requirements beyond IEC 559	43
Annex E (informative) Bindings for specific languages	44
E.1 General comments	44
E.2 Ada	45
E.3 BASIC	47
E.4 C	49
E.5 Common Lisp	52
E.6 Fortran	55
E.7 Modula-2	57
E.8 Pascal and Extended Pascal	57
E.9 PL I	60
Annex F (informative) Example of a conformity statement	62
F.1 Types	63
F.2 Integer parameters	63
F.3 Floating point parameters	63
F.4 Definitions	63
F.5 Expressions	64

	Page
F.6 Notification	64
Annex G (informative) Example programs	65
G.1 Verifying platform acceptability	65
G.2 Selecting alternate code	65
G.3 Terminating a loop	65
G.4 Fast versus reliable	66
G.5 High-precision multiply	66
G.6 Estimating error	66
G.7 Saving and restoring indicators	67
Annex H (informative) Bibliography	67
Annex J (informative) Glossary	69

Descriptors: Data processing, computer software, computer interfaces, programming languages, arithmetics, arithmetical operation.

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.

Annex A to Annex 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.

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.

NOTE 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.

NOTE 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, Z denotes the set of mathematical integers, R denotes the set of real numbers, and C denotes the set of complex numbers. Note that $Z \subset R \subset C$.

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 $\Leftrightarrow$ 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 $x \in R$, the notation $\lfloor x \rfloor$ designates the largest integer not greater than x :

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

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 Z , R , or 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.)

NOTE 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.

NOTE 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, sub-routine 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 $rnd : R \rightarrow X$ (where X is a discrete subset of R) that maps each element of X to itself, and is monotonic non-decreasing. Formally, if x and y are in R ,

$$x \in X \implies rnd(x) = x$$

$$x < y \implies rnd(x) \leq rnd(y)$$

note that if $u \in R$ 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 $u \in R$ 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 $u \in R$ 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 $u \in R$ 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

$add_I I \times I \rightarrow I \cup \{\mathbf{integer_overflow}\}$

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**.

NOTE 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.

NOTE 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 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

$$I = Z$$

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

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

and $minint$ and $maxint$ shall satisfy

$$\begin{aligned} maxint &> 0 \\ \text{and one of: } minint &= 0 \\ minint &= -(maxint) \\ minint &= -(maxint + 1) \end{aligned}$$

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*.

NOTE 1 Most traditional programming languages call for bounded integers. Others allow an integer type to have an unbounded range. A few languages permit the implementation to decide whether an integer type will be bounded or unbounded. (See A.5.1.0.3 for further discussion.)

NOTE 2 Operations on unbounded integers will not overflow, but may fail due to exhaustion of resources.

An implementation may provide more than one integer type. A method shall be provided for a program to obtain the values of the parameters $bounded$, $modulo$, $minint$, and $maxint$, for each integer type provided.

NOTE 3 If the value of a parameter (like $bounded$) is dictated by a language standard, implementations of that language need not provide program access to that parameter explicitly.

5.1.1 Operations

For each integer type, the following operations shall be provided:

$add_I: I \times I \rightarrow I \cup \{\text{integer_overflow}\}$
$sub_I: I \times I \rightarrow I \cup \{\text{integer_overflow}\}$
$mul_I: I \times I \rightarrow I \cup \{\text{integer_overflow}\}$
$div_I: I \times I \rightarrow I \cup \{\text{integer_overflow, undefined}\}$
$rem_I: I \times I \rightarrow I \cup \{\text{undefined}\}$
$mod_I: I \times I \rightarrow I \cup \{\text{undefined}\}$
$neg_I: I \rightarrow I \cup \{\text{integer_overflow}\}$
$abs_I: I \rightarrow I \cup \{\text{integer_overflow}\}$
$sign_I: I \rightarrow I$
$eq_I: I \times I \rightarrow Boolean$
$neq_I: I \times I \rightarrow Boolean$
$lss_I: I \times I \rightarrow Boolean$
$leq_I: I \times I \rightarrow Boolean$

$$gtr_I: I \times I \rightarrow \text{Boolean}$$

$$geq_I: I \times I \rightarrow \text{Boolean}$$

If I is unsigned, it is permissible to omit the operations neg_I , abs_I , and $sign_I$.

5.1.2 Modulo integers versus overflow

If *bounded* is **true**, the mathematical operations $+$, $-$, $*$, and $/$ (after rounding) can produce results that lie outside the set I . In such cases, the computational operations add_I , sub_I , mul_I , and div_I shall either cause a notification (if *modulo* = **false**), or return a “wrapped” result (if *modulo* = **true**).

The helper function

$$wrap_I: Z \rightarrow I$$

(which produces the wrapped result) is defined as follows:

$$wrap_I(x) = x + j * (maxint - minint + 1) \quad \text{for some } j \text{ in } Z$$

$$wrap_I(x) \in I$$

5.1.3 Axioms

For all values x and y in I , the following shall apply:

$add_I(x, y)$	$= x + y$	if $x + y \in I$
	$= wrap_I(x + y)$	if $x + y \notin I$ and <i>modulo</i> = true
	$= \text{integer_overflow}$	if $x + y \notin I$ and <i>modulo</i> = false
$sub_I(x, y)$	$= x - y$	if $x - y \in I$
	$= wrap_I(x - y)$	if $x - y \notin I$ and <i>modulo</i> = true
	$= \text{integer_overflow}$	if $x - y \notin I$ and <i>modulo</i> = false
$mul_I(x, y)$	$= x * y$	if $x * y \in I$
	$= wrap_I(x * y)$	if $x * y \notin I$ and <i>modulo</i> = true
	$= \text{integer_overflow}$	if $x * y \notin I$ and <i>modulo</i> = false

An implementation shall provide one or both of the operation pairs div_I^f/rem_I^f and div_I^t/rem_I^t .

(The functions $\lfloor \rfloor$ and $tr()$ are defined in 4.1.)

$div_I^f(x, y)$	$= \lfloor x/y \rfloor$	if $y \neq 0$ and $\lfloor x/y \rfloor \in I$
	$= wrap_I(\lfloor x/y \rfloor)$	if $y \neq 0$ and $\lfloor x/y \rfloor \notin I$ and <i>modulo</i> = true
	$= \text{integer_overflow}$	if $y \neq 0$ and $\lfloor x/y \rfloor \notin I$ and <i>modulo</i> = false
	$= \text{undefined}$	if $y = 0$
$rem_I^f(x, y)$	$= x - (\lfloor x/y \rfloor * y)$	if $y \neq 0$
	$= \text{undefined}$	if $y = 0$
$div_I^t(x, y)$	$= tr(x/y)$	if $y \neq 0$ and $tr(x/y) \in I$
	$= wrap_I(tr(x/y))$	if $y \neq 0$ and $tr(x/y) \notin I$ and <i>modulo</i> = true
	$= \text{Integer_overflow}$	if $y \neq 0$ and $tr(x/y) \notin I$ and <i>modulo</i> = false
	$= \text{undefined}$	if $y = 0$
$rem_I^t(x, y)$	$= x - (tr(x/y) * y)$	if $y \neq 0$
	$= \text{undefined}$	if $y = 0$

An implementation shall provide one or both of mod_I^a and mod_I^p .

$mod_I^a(x, y)$	$= (\lfloor x/y \rfloor * y)$	if $y \neq 0$
	$= \text{undefined}$	if $y = 0$

$mod_I^p(x, y)$	$= (\lfloor x/y \rfloor * y)$	if $y > 0$
	$= \text{undefined}$	if $y \leq 0$
$neg_I(x)$	$= -x$	if $-x \in I$
	$= wrap_I(-x)$	if $-x \notin I$ and $modulo = \text{true}$
	$= \text{integer_overflow}$	if $-x \notin I$ and $modulo = \text{false}$
$abs_I(x)$	$= x $	if $ x \in I$
	$= wrap_I(x)$	if $ x \notin I$ and $modulo = \text{true}$
	$= \text{integer_overflow}$	if $ x \notin I$ and $modulo = \text{false}$
$sign_I(x)$	$= 1$	if $x > 0$
	$= 0$	if $x = 0$
	$= -1$	if $x < 0$
	$eq_I(x, y) = \text{true} \iff x = y$	
	$neq_I(x, y) = \text{true} \iff x \neq y$	
	$lss_I(x, y) = \text{true} \iff x < y$	
	$leq_I(x, y) = \text{true} \iff x \leq y$	
	$gtr_I(x, y) = \text{true} \iff x > y$	
	$geq_I(x, y) = \text{true} \iff x \geq y$	

5.2 Floating point types

NOTE 1 This part of ISO/IEC 10967 does not advocate any particular representation for floating point values. However, concepts such as *radix*, *precision*, and *exponent* are derived from an abstract model of such values as discussed in A.5.2.

A floating point type F shall be finite subset of R , characterized by five parameters:

$r \in \mathbb{Z}$	(the radix of F)
$p \in \mathbb{Z}$	(the precision of F)
$emin \in \mathbb{Z}$	(the smallest exponent of F)
$emax \in \mathbb{Z}$	(the largest exponent of F)
$denorm \in \text{Boolean}$	(whether F contains denormalized values)

The parameters r and p shall satisfy

$$r \geq 2 \text{ and } p \geq 2$$

and r should be even.

The parameters $emin$, $emax$, and p shall satisfy

$$p - 2 \leq -emin \leq r^p - 1$$

$$p \leq emax \leq r^p - 1$$

Given specific values for r , p , $emin$, $emax$, and $denorm$, the following sets are defined:

$$F_N = \{0, \pm i * r^{e-p} \mid i, e \in \mathbb{Z}, r^{p-1} \leq i \leq r^p - 1, emin \leq e \leq emax\}$$

$$F_D = \{\pm i * r^{e-p} \mid i, e \in \mathbb{Z}, 1 \leq i \leq r^{p-1} - 1, e = emin\}$$

$$F = F_N \cup F_D \quad \text{if } denorm = \text{true}$$

$$= F_N \quad \text{if } denorm = \text{false}$$

The elements of F_N are called *normalized* floating point values because of the constraint $r^{p-1} \leq i \leq r^p - 1$. The elements of F_D are called *denormalized* floating point values.

NOTE 2 The terms *normalized* and *denormalized* refer to the mathematical values involved, not to any method of representation.

The type F is called *normalized* if it contains only normalized values, and called *denormalized* if it contains denormalized values as well.

An implementation may provide more than one floating point data type. A method shall be provided for a program to obtain the values of the parameters r , p , $emin$, $emax$, and $denorm$, for each floating point type provided.

NOTE 3 The conditions placed upon the parameters r , p , $emin$, and $emax$ are sufficient to guarantee that the abstract model of F is well-defined and contains its own parameters. More stringent conditions are needed to produce a computationally useful floating point type. These are design decisions which are beyond the scope of this part of ISO/IEC 10967. (See A.5.2.)

The following set is an unbounded extension of F :

$$F^* = F_N \cup F_D \cup \{ \pm i * r^{e-p} \mid i, e \in \mathbb{Z}, r^{p-1} \leq i \leq r^p - 1, e > emax \}$$

NOTE 4 The set F^* contains values of magnitude larger than those that are representable in the type F . F^* will be used in defining rounding.

5.2.1 Range and granularity constants

The range and granularity of F are characterized by the following derived constants:

$$\begin{aligned} fmax &= \max \{z \in F \mid z > 0\} &= (1 - r^{-p}) * r^{emax} \\ fmin_N &= \min \{z \in F_N \mid z > 0\} &= r^{emin-1} \\ fmin_D &= \min \{z \in F_D \mid z > 0\} &= r^{emin-p} \\ fmin &= \min \{z \in F \mid z > 0\} &= \begin{cases} fmin_D & \text{if } denorm = \text{true} \\ fmin_N & \text{if } denorm = \text{false} \end{cases} \\ epsilon &= r^{1-p} & \text{(the maximum relative spacing in } F_N \end{aligned}$$

A method shall be provided for a program to obtain the values of the derived constants $fmax$, $fmin$, $fmin_N$, and $epsilon$, for each floating point data type provided.

5.2.2 Operations

For each floating point type, the following operations shall be provided:

add_F :	$F \times F \rightarrow F \cup \{\text{floating_overflow, underflow}\}$
sub_F :	$F \times F \rightarrow F \cup \{\text{floating_overflow, underflow}\}$
mul_F :	$F \times F \rightarrow F \cup \{\text{floating_overflow, underflow}\}$
div_F :	$F \times F \rightarrow F \cup \{\text{floating_overflow, underflow, undefined}\}$
neg_F :	$F \rightarrow F$
abs_F :	$F \rightarrow F$
$sign_F$:	$F \rightarrow F$
$exponent_F$:	$F \rightarrow J \cup \{\text{undefined}\}$
$fraction_F$:	$F \rightarrow F$
$scale_F$:	$F \times J \rightarrow F \cup \{\text{floating_overflow, underflow}\}$
$succ_F$:	$F \rightarrow F \cup \{\text{floating_overflow}\}$
$pred_F$:	$F \rightarrow F \cup \{\text{floating_overflow}\}$
ulp_F :	$F \rightarrow F \cup \{\text{underflow, undefined}\}$
$trunc_F$:	$F \times J \rightarrow F$
$round_F$:	$F \times J \rightarrow F \cup \{\text{floating_overflow}\}$
$intpart_F$:	$F \rightarrow F$
$fractpart_F$:	$F \rightarrow F$
eq_F :	$F \times F \rightarrow \text{Boolean}$
neq_F :	$F \times F \rightarrow \text{Boolean}$
lss_F :	$F \times F \rightarrow \text{Boolean}$
leq_F :	$F \times F \rightarrow \text{Boolean}$
gtr_F :	$F \times F \rightarrow \text{Boolean}$
geq_F :	$F \times F \rightarrow \text{Boolean}$

J shall be a data type containing all integer values from $-(emax - emin + p - 1)$ to $(emax - emin + p - 1)$ inclusive.

NOTE 1 J should be a conforming integer type (if practical). However, a floating point type will suffice.

NOTE 2 Operations are permitted to accept inputs not listed above. In particular, IEC 559 requires floating point operations to accept infinities and NaNs as inputs. Such values are not in F .

5.2.3 Approximate operations

The operations add_F , sub_F , mul_F , div_F , and $scale_F$ are approximations of exact mathematical operations. They differ from their exact counterparts in that

- a) they produce “rounded” results, and
- b) they produce notifications.

The axioms for floating point define these approximate operations *as if* they were computed in three stages:

- a) Compute the exact mathematical answer. (For addition and subtraction, a close approximation to the exact answer may be used.)
- b) Round this answer to p digits of precision. (The precision will be less if the answer is in the denormalized range.)
- c) Determine if notification is required.

These stages will be modelled by three helper functions: add_F^* (stage a, for addition only), rnd_F (stage b), and $result_F$ (stage c). These helper functions are not visible to the programmer, and are not required to be part of the implementation. An actual implementation need not perform the above stages at all, merely return a result (or produce a notification) as if it had.

Different floating point types may have different versions of add_F^* , rnd_F , and $result_F$.

5.2.4 Approximate addition

Some hardware implementations of addition compute an approximation to addition that loses information prior to rounding. As a consequence, $x + y = u + v$ may not imply $add_F(x, y) = add_F(u, v)$.

The add_F^* helper function is introduced to model this pre-rounding approximation:

$$add_F^*: F \times F \rightarrow R$$

For all values u, v, x , and y in F , and i in Z , the following axioms are satisfied by add_F^* :

$$add_F^*(u, v) = add_F^*(v, u)$$

$$add_F^*(-u, -v) = - add_F^*(u, v)$$

$$x \leq u + v \leq y \implies x \leq add_F^*(u, v) \leq y$$

$$u \leq v \implies add_F^*(u, x) \leq add_F^*(v, x)$$

If $u, v, u * r^i$, and $v * r^i$ are all in F_N ,

$$add_F^*(u * r^i, v * r^i) = add_F^*(u, v) * r^i$$

NOTE The above five axioms capture the following properties:

- a) Add_F^* is commutative.
- b) Add_F^* is sign symmetric.
- c) Add_F^* is in the same “basic interval” as $u + v$, and is exact if $u + v$ is exactly representable. (A basic interval is the range between two adjacent F values.)
- d) Add_F^* is monotonic.
- e) Add_F^* does not depend on the exponents of its arguments (only on the difference of the exponents).

Ideally, no information should be lost before rounding. Thus, add_F^* should satisfy

$$add_F^*(x, y) = x + y$$

5.2.5 Rounding

For floating point operations, rounding is the process of taking an exact result in R and producing a p -digit approximation.

The rnd_F helper function is introduced to model this process:

$$rnd_F: R \rightarrow F^*$$

Rnd_F is a rounding function as defined in 4.2. Rnd_F is sign symmetric. That is, for $x \in R$,

$$rnd_F(-x) = -rnd_F(x)$$

For $x \in R$ and $i \in Z$, such that $|x| \geq fmin_N$ and $|x * r^i| \geq fmin_N$,

$$rnd_F(x * r^i) = rnd_F(x) * r^i$$

NOTE This rule means that the rounding function does not depend on the “exponent” part of the real number *except* when denormalization occurs.

If, for some $x \in R$ and some $i \in Z$, such that $|x| < fmin_N$ and $|x * r^i| \geq fmin_N$, the formula

$$rnd_F(x * r^i) = rnd_F(x) * r^i$$

does *not* hold, then rnd_F is said to have a *denormalization loss* at x .

5.2.6 Result function

A floating point operation produces a rounded result or a notification. The decision is based on the computed result (either before or after rounding).

The $result_F$ helper function is introduced to model this decision:

$$result_F: R \times (R \rightarrow F^*) \rightarrow F \cup \{\text{floating_overflow}, \text{underflow}\}$$

NOTE The first input to $result_F$ is the computed result before rounding, and the second input is the rounding function to be used.

For all values x in R , and any rounding function rnd in $(R \rightarrow F^*)$, the following shall apply:

For $x = 0$ or $fmin_N \leq |x| \leq fmax$:

$$result_F(x, rnd) = rnd(x)$$

For $|x| > fmax$:

$$\begin{aligned} result_F(x, rnd) &= rnd(x) && \text{if } |rnd(x)| = fmax \\ &= \text{floating_overflow} && \text{otherwise} \end{aligned}$$

For $0 < |x| < fmin_N$:

$$\begin{aligned} result_F(x, rnd) &= rnd(x) \text{ or } \text{underflow} && \text{if } |rnd(x)| = fmin_N \\ &= rnd(x) \text{ or } \text{underflow} && \text{if } |rnd(x)| \in F_D, \text{denorm} = \text{true}, \text{ and } rnd \text{ has no} \\ & && \text{denormalization loss at } x \\ &= \text{underflow} && \text{otherwise} \end{aligned}$$

An implementation is allowed to choose between $rnd(x)$ and **underflow** in the region between 0 and $fmin_N$. However, a denormalized value for $rnd(x)$ can be chosen only if *denorm* is **true** and no denormalization loss occurs at x . An implementation shall document how the choice between $rnd(x)$ and **underflow** is made.

5.2.7 Axioms

For convenience, define two helper functions: e_F and rn_F .

Define $e_F: R \rightarrow Z$ such that

$$\begin{aligned} e_F(x) &= \lfloor \log |x| \rfloor + 1 && \text{if } |x| \geq fmin_N \\ &= emin && \text{if } |x| < fmin_N \end{aligned}$$

NOTE 1 The value $e_F(x)$ is that of the e in the definitions of F_N , F_D , and F^* , and has been defined to have the value *emin* at 0. When x is in F_D , $e_F(x)$ is *emin* regardless of x .

Define $rn_F: F \times Z \rightarrow F^*$ such that

$$rn_F(x, n) = sign_F(x) * \lfloor |x| / r^{e_F(x) - n} + 1/2 \rfloor * r^{e_F(x) - n}$$

NOTE 2 The value $rn_F(x)$ is rounded to n digits of precision (using traditional round to nearest in which ties round away from zero).

For all values x and y in F , and n an integer in J the following shall apply:

$$\begin{aligned} add_F(x, y) &= result_F(add_F^*(x, y), rnd_F) \\ sub_F(x, y) &= add_F(x, -y) \\ mul_F(x, y) &= result_F(x * y, rnd_F) \\ div_F(x, y) &= result_F(x/y, rnd_F) && \text{if } y \neq 0 \\ &= \text{undefined} && \text{if } y = 0 \end{aligned}$$

$neg_F(x)$	$= -x$	
$abs_F(x)$	$= x $	
$sign_F(x)$	$= 1$	if $x > 0$
	$= 0$	if $x = 0$
	$= -1$	if $x < 0$
$exponent_F(x)$	$= \lfloor \log_r x \rfloor + 1$	if $x \neq 0$
	$= \text{undefined}$	if $x = 0$
$fraction_F(x)$	$= x/r^{exponent_F(x)}$	if $x \neq 0$
	$= 0$	if $x = 0$
$scale_F(x, n)$	$= result_F\{x * r^n, rnd_F\}$	
$succ_F(x)$	$= \min\{z \in F \mid z > x\}$	if $x \neq fmax$
	$= \text{floating_overflow}$	if $x = fmax$
$pred_F(x)$	$= \max\{z \in F \mid z < x\}$	if $x \neq -fmax$
	$= \text{floating_overflow}$	if $x = -fmax$
$ulp_F(x)$	$= r^{e_F(x)-p}$	if $x \neq 0$ and $r^{e_F(x)-p} \in F$
	$= \text{underflow}$	if $x \neq 0$ and $r^{e_F(x)-p} \notin F$
	$= \text{undefined}$	if $x = 0$
$trunc_F(x, n)$	$= \lfloor x/r^{e_F(x)-n} \rfloor * r^{e_F(x)-n}$	if $x \geq 0$
	$= trunc_F(-x, n)$	if $x < 0$
$round_F(x, n)$	$= rn_F(x, n)$	if $ rn_F(x, n) \leq fmax$
	$= \text{floating_overflow}$	if $ rn_F(x, n) > fmax$
$intpart_F(x)$	$= sign_F(x) * \lfloor x \rfloor$	
$fractpart_F(x)$	$= x - intpart_F(x)$	
$eq_F(x, y) = \text{true}$	$\iff x = y$	
$neq_F(x, y) = \text{true}$	$\iff x \neq y$	
$lss_F(x, y) = \text{true}$	$\iff x < y$	
$leq_F(x, y) = \text{true}$	$\iff x \leq y$	
$gtr_F(x, y) = \text{true}$	$\iff x > y$	
$geq_F(x, y) = \text{true}$	$\iff x \geq y$	

5.2.8 Rounding constants

Two derived constants shall be provided to characterize the rounding function: rnd_error and rnd_style . Define the derived constant rnd_error to be the smallest element of F such that

$$|x - rnd_F(x)| \leq rnd_error * r^{e_F[rnd_F(x)]-p}$$

for all $x \in R$. If $add_F^*(x, y)$ is not identically equal to $x + y$ for all $x, y \in F$, then rnd_error shall be defined as 1. (See add_F^* in 5.2.4.)

NOTE The requirement that rnd_F be a rounding function implies that $rnd_error \leq 1$. However, some definitions for rnd_F may yield smaller values for rnd_error . (See A.5.2.8.)

A method shall be provided for a program to obtain the value of the derived constant rnd_error for each floating point data type provided.

Rnd_F has the *round toward zero* property if for $x \in R$

$$|rnd_F(x)| \leq |x|$$

Rnd_F has the *round to nearest* property if for $x \in R$

$$|rnd_F(x) - x| \leq \frac{1}{2} * r^{e_F(x)-p}$$

Note that the behavior when x is exactly halfway between values in F is not specified by this definition.

The derived constant rnd_style , having one of three allowed constant values, is defined by

rnd_style	=	nearest	if rnd_F has the round to nearest property
	=	truncate	if rnd_F has the round toward zero property
	=	other	otherwise

If $add_F^*(x, y)$ is not identically equal to $x + y$ for all $x, y \in F$, then rnd_style shall be defined as **other**.

A method shall be provided for a program to obtain the value of the derived constant rnd_style for each floating point data type provided. In addition, a notation for each of the values **nearest**, **truncate**, and **other** shall be provided such that the value of rnd_style can be compared to the constants.

5.2.9 Conformity to IEC 559

One further behavioral parameter shall be provided for each floating point type F :

$iec_559 \in Boolean$ (whether F conforms to IEC 559)

The parameter iec_559 shall be true only when the type F completely conforms to the requirements of IEC 559 (also known as IEEE 754). F may correspond to any of the floating point types defined in IEC 559.

When iec_559 is **true**, all the facilities required by IEC 559 shall be provided. Methods shall be provided for a program to access each such facility. In addition, documentation shall be provided to describe these methods, and all implementation choices.

NOTE 1 The IEC 559 facilities include values for infinities and NaNs, extended comparisons, program control of rounding, an inexact exception flag, and so on. See Annex C for more information.

When iec_559 is **true**, all operations and values common to this part of ISO/IEC 10967 and IEC 559 shall satisfy the requirements of both standards. Values present only in IEC 559 (-0 , $+\infty$, $-\infty$, and the NaNs) need only satisfy the requirements of IEC 559. The value set F as used in the definitions and axioms of 5.2 does not contain these extra values — it only contains the common values. Thus, the axioms in 5.2.7 apply only to the common values. Nevertheless, the operations in 5.2.2 shall not distinguish -0 from $+0$.

NOTE 2 An implementation of IEC 559 can distinguish -0 from $+0$ only by the use of operations not in 5.2.2, or by the generation or use of values not in F (infinities).

A method shall be provided for a program to obtain the value of the parameter iec_559 for each floating point data type provided.

5.3 Conversion operations

A conversion operation is a function from one arithmetic type to another arithmetic type. Conversion operations shall be provided

- between any two distinct integer types,
- between any two distinct floating point types of the same radix,
- from any integer type to any floating point type, and
- from any floating point type to any integer type.

NOTE 1 This part of ISO/IEC 10967 does not define conversion operations between floating point types with different radices.

Let I_a and I_b be two integer types. The conversion operation

$$cvt_{I_a \rightarrow I_b}: I_a \rightarrow I_b \cup \{\mathbf{integer_overflow}\}$$

shall be defined by

$$\begin{aligned} cvt_{I_a \rightarrow I_b}(x) &= x && \text{if } x \in I_b \\ &= wrap_{I_b}(x) && \text{if } x \notin I_b \text{ and } modulo_{I_b} = \mathbf{true} \\ &= \mathbf{integer_overflow} && \text{if } x \notin I_b \text{ and } modulo_{I_b} = \mathbf{false} \end{aligned}$$

where $modulo_{I_b}$ is the *modulo* parameter for type I_b .

Let $nearest_X$ be a helper rounding function from R to X satisfying the round to nearest property.

Let F_a and F_b be two floating point types with the same radix. The conversion operation

$$cvt_{F_a \rightarrow F_b}: F_a \rightarrow F_b \cup \{\mathbf{floating_overflow}, \mathbf{underflow}\}$$

shall be defined by

$$cvt_{F_a \rightarrow F_b}(x) = result_{F_b}(x, nearest_{F_b})$$

Let I be an integer type, and F be a floating point type. The conversion operation

$$cvt_{I \rightarrow F}: I \rightarrow F \cup \{\mathbf{floating_overflow}\}$$

shall be defined by

$$cvt_{I \rightarrow F}(x) = result_F(x, nearest_F)$$

Let F be a floating point type, and I be an integer type. The conversion operation

$$cvt_{F \rightarrow I}: F \rightarrow I \cup \{\mathbf{integer_overflow}\}$$

shall be defined by

$$\begin{aligned} cvt_{F \rightarrow I}(x) &= rnd_{F \rightarrow I}(x) && \text{if } rnd_{F \rightarrow I}(x) \in I \\ &= wrap_I(rnd_{F \rightarrow I}(x)) && \text{if } rnd_{F \rightarrow I}(x) \notin I \text{ and } modulo_I = \mathbf{true} \\ &= \mathbf{integer_overflow} && \text{if } rnd_{F \rightarrow I}(x) \notin I \text{ and } modulo_I = \mathbf{false} \end{aligned}$$

where $modulo_I$ is the *modulo* parameter for type I and $rnd_{F \rightarrow I}$ is a helper rounding function from R to Z .

NOTE 2 With proper choice of $rnd_{F \rightarrow I}$, the function $cvt_{F \rightarrow I}$ could be identical with the function *floor*, *truncate*, *round*, or *ceiling*. These functions will be described in more detail in Part 2 of ISO/IEC 10967.

An implementation may provide more than one conversion operation for a given pair of types. In particular, different choices for $rnd_{F \rightarrow I}$ or $nearest_F$ can produce different conversion operations.

6 Notification

Notification is the process by which a user or program is informed that an arithmetic operation cannot be performed. Specifically, a notification shall occur when any arithmetic operation returns an exceptional value as defined in clause 5.

6.1 Notification alternatives

Three alternatives for notification are provided here. The requirements are:

- The alternative in 6.1.1 shall be supplied in conjunction with any language which provides support for exception handling.
- The alternative in 6.1.2 shall be supplied in the absence of language support for exception handling.
- The alternative in 6.1.3 shall be supplied by all implementations.

6.1.1 Language defined notification

If the programming language in use defines an exception handling mechanism that can

- detect the occurrence of arithmetic exceptions,
- report such exceptions to the executing program,
- permit the programmer to specify code to compensate for such exceptions, and then
- continue program execution,

then notifications shall be handled by that language defined mechanism.

Such a mechanism may be defined as part of the programming language standard itself or by a separate binding standard.

NOTE The exception handling mechanisms of Ada and PL/I are examples of language defined notification. In these languages, an exception causes a prompt alteration of control flow to execute user provided exception handling code. Other notification mechanisms, such as continued execution with special non-numeric "error values," may be appropriate for other languages.

The manner in which the exception handling code is specified and the capabilities of such exception handling code (including whether it is possible to resume the operation which caused the notification) is the province of the language standard, not this arithmetic standard.

If no exception handling code is provided for a particular occurrence of the return of an exceptional value as defined in clause 5, that fact shall be reported to the user of that program in an unambiguous and "hard to ignore" manner. (See 6.1.3.)