TECHNICAL

ISO/IEC

24732

First edition
2009-01-15

Information technology — Programming languages, their environments and system software interfaces — Extension for the programming language C to support decimal floating-point arithmetic

Technologies de l'information — Langages de programmation, leur environnement et interfaces des logiciels de systèmes — Extension pour que le langage de programmation C supporte l'arithmétique du point flottant décimal

Reference number
ISO/IEC TR 24732:2009(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. 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.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 24732, which is a Technical Report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

0 Introduction

0.1 Background

Most of today's general purpose computing architectures provide binary floating-point arithmetic in hardware. Binary floating-point is an efficient representation which minimizes memory use, and is simpler to implement than floating-point arithmetic using other bases. It has therefore become the norm for scientific computations, with almost all implementations following the IEEE 754 standard for binary floating-point arithmetic.

However, human computation and communication of numeric values almost always uses decimal arithmetic and decimal notations. Laboratory notes, scientific papers, legal documents, business reports and financial statements all record numeric values in decimal form. When numeric data are given to a program or are displayed to a user, binary to-and-from decimal conversion is required. There are inherent rounding errors involved in such conversions; decimal fractions cannot, in general, be represented exactly by binary floating-point values. These errors often cause usability and efficiency problems, depending on the application.

These problems are minor when the application domain accepts, or requires results to have, associated error estimates (as is the case with scientific applications). However, in business and financial applications, computations are either required to be exact (with no rounding errors) unless explicitly rounded, or be supported by detailed analyses that are auditable to be correct. Such applications therefore have to take special care in handling any rounding errors introduced by the computations.

The most efficient way to avoid conversion error is to use decimal arithmetic. Currently, the IBM zArchitecture (and its predecessors since System/360) is a widely used system that supports built-in decimal arithmetic. This, however, provides integer arithmetic only, meaning that every number and computation has to have separate scale information preserved and computed in order to maintain the required precision and value range. Such scaling is difficult to code and is error-prone; it affects execution time significantly, and the resulting program is often difficult to maintain and enhance.

Even though the hardware may not provide decimal arithmetic operations, the support can still be emulated by software. Programming languages used for business applications either have native decimal types (such as PL/I, COBOL, C#, or Visual Basic) or provide decimal arithmetic libraries (such as the BigDecimal class in Java). The arithmetic used in business applications, nowadays, is almost invariably decimal floating-point; the COBOL 2002 ISO standard, for example, requires that all standard decimal arithmetic calculations use 32-digit decimal floating-point.

Arguably, the C language hits a sweet spot within the wide range of programming languages available today – it strikes an optimal balance between usability and performance. Its simple and expressive syntax makes it easy to program; and its close-to-the-hardware semantics makes it efficient. Despite the advent of newer programming languages, C is still often used together with other languages to code the computationally intensive part of an application. In many cases, entire business applications are written in C/C++. To maintain the vitality of C, the need for decimal arithmetic by the business and financial community cannot be ignored.

The importance of this has been recognized by the IEEE. The IEEE 754 standard is currently being revised, and the major change in that revision is the addition of decimal floating-point formats and arithmetic.

Historically there has been a close tie between IEEE 754 and C with respect to floating-point specification. This Technical Report proposes to add decimal floating types and arithmetic to the C programming language specification.

0.2 The arithmetic model

This Technical Report proposes to add support for the decimal formats for floating-point data specified in IEEE 754-2008, with operations and behaviors consistent with that specification. IEEE 754-2008 provides a unified specification for floating-point arithmetic using both binary radix and decimal radix representations. For binary radix, it specifies upwardly-compatible extensions to the previous version, IEEE 754-1985 (equivalently IEC 60559:1989, which is already supported by C99 implementations that define the macro **__STDC_IEC_559__**).

Those extensions are not considered in this proposal. Instead, this proposal confines itself to supporting the decimal radix formats, which are new in this revision of IEEE 754.

The model of floating-point arithmetic used in IEEE 754-2008 has three components:

— *data* - numbers and NaNs, which can be manipulated by, or be the results of, the operations it specifies

— *operations* - (addition, multiplication, conversions, etc) which can be carried out on data

— *context* - the status of operations (namely, exceptions flags), and controls to govern the results of operations (for example, rounding modes). (IEEE 754-2008 does not use a single term to refer to these collectively.)

The model defines these components in the abstract. It neither defines the way in which operations are expressed (which might vary depending on the computer language or other interface being used), nor does it define the concrete representation (specific layout in storage, or in a processor's register, for example) of data or context, except that it does define specific encodings that are to be used for data that may be exchanged between different implementations that conform to the specification.

From the perspective of the C language, *data* are represented by data types, *operations* are defined within expressions, and *context* is the floating environment specified in **<fenv.h>**. This Technical Report specifies how the C language implements these components.

0.3 The formats

IEEE 754-2008 specifies *formats*, in terms of their radix, exponent range, and precision (significand length), to support general purpose decimal floating-point arithmetic. It specifies operation semantics in terms of values and abstract representations of data (format members). It also specifies bit-level encodings for formats intended for data interchange.

C99 specifies floating-point arithmetic using a two-layer organization. The first layer provides a specification using an abstract model. The representation of a floating-point number is specified in an abstract form where the constituent components of the representation are defined (sign, exponent, significand) but not the internals of these components. In particular, the exponent range, significand size, and the base (or radix) are implementation defined. This allows flexibility for an implementation to take advantage of its underlying hardware architecture. Furthermore, certain behaviors of operations are also implementation defined, for example in the area of handling of special numbers and in exceptions.

The reason for this approach is historical. At the time when C was first standardized, there were already various hardware implementations of floating-point arithmetic in common use. Specifying the exact details of a representation would make most of the existing implementations at the time not conforming.

C99 provides a binding to IEEE 754 by specifying an Annex F, *IEC 60559 floating point arithmetic*, and adopting that standard by reference. An implementation may choose not to conform to IEEE 754 and indicates that by not defining the macro **__STDC_IEC_559__**.This means not all implementations need to support IEEE 754, and the floating-point arithmetic need not be binary.

This Technical Report specifies decimal floating-point arithmetic according to IEEE 754-2008, with the constituent components of the representation defined. This is more stringent than the existing C99 approach for the floating types. Since it is expected that all decimal floating-point hardware implementations will conform to the revised IEEE 754, binding to this standard directly benefits both implementers and programmers.