The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.3.1.5 Real floating types

float promoted to double or long double

When a **float** is promoted to **double** or **long double**, or a **double** is promoted to **long double**, its value is unchanged (if the source value is represented in the precision and range of its type). 695

Commentary

Although not worded as such, this is a requirement on the implementation. The type **double** must have at least the same precision in the significand and at least as much range in the exponent as the type **float**, similarly for the types **double** and **long double**.

FLT_EVAL_METHOD

A situation where the source value might not be represented in the precision and range of its type is when FLT_EVAL_METHOD has a non-zero value. For instance, if FLT_EVAL_METHOD has a value of 2, then the value 0.1f is represented to the precision of **long double**, while the type remains as **float**. If a cast to **double** is performed<footnote, 87> the value may be different to that obtained when FLT_EVAL_METHOD was zero.

The wording was changed by the response to DR #318.

C++

3.9.1p8

> *The type **double** provides at least as much precision as **float**, and the type **long double** provides at least as much precision as **double**. The set of values of the type **float** is a subset of the set of values of the type **double**; the set of values of the type **double** is a subset of the set of values of the type **long double**.*

This only gives a relative ordering on the available precision. It does not say anything about promotion leaving a value unchanged.

4.6p1

> *An rvalue of type **float** can be converted to an rvalue of type **double**. The value is unchanged.*

There is no equivalent statement for type **double** to **long double** promotions. But there is a general statement about conversion of floating-point values:

4.8p1

> *An rvalue of floating point type can be converted to an rvalue of another floating point type. If the source value can be exactly represented in the destination type, the result of the conversion is that exact representation.*

Given that (1) the set of values representable in a floating-point type is a subset of those supported by a wider floating-point type (3.9.1p8); and (2) when a value is converted to the wider type, the exact representation is required to be used (by 4.8p1)— the value must be unchanged.

Other Languages

Some languages specify one floating type, while others recognize that processors often support several different floating-point precisions and define mechanisms to allow developers to specify different floating types. While implementations that provide multiple floating-point types usually make use of the same processor hardware as that available to a C translator, other languages rarely contain this requirement.

Common Implementations

This requirement is supported by IEC 60559, where each representation is an extension of the previous ones holding less precision.

double demoted to another floating type

When a **double** is demoted to **float**, a **long double** is demoted to **double** or **float**, or a value being represented in greater precision and range than required by its semantic type (see 6.3.1.8) is explicitly converted <iso_delete>to its semantic type</iso_delete> (including to its own type), if the value being converted can be represented exactly in the new type, it is unchanged. 696

Commentary

A simple calculation would suggest that unless an implementation uses the same representation for floating-point types, the statistically likelihood of a demoted value being exactly representable in the new type would be very low (an IEC 60559 double-precision type contains 2^{32} values that convert to the same single-precision value). However, measurements of floating-point values created during program execution show a surprisingly high percentage of value reuse (these results are discussed elsewhere).

binary * result
value profiling

A situation where a value might be represented in greater precision and range than required by its type is when FLT_EVAL_METHOD has a non-zero value.

FLT_EVAL_METH
695 float promoted to double or long double

The wording was changed by the response to DR #318.

C90

This case is not specified in the C90 Standard.

Coding Guidelines

Relying on the converted value being equal to the original value is simply a special case of comparing two floating-point values for equality.

?? equality operators not floating-point operands

Why is a floating-point value being demoted? If a developer is concerned about a floating-point value being represented to a greater precision than required by its semantic type, explicit conversions might be used simply as a way of ensuring known behavior in the steps in a calculation. Or perhaps a computed value is being assigned to an object. There are sometimes advantages to carrying out the intermediate steps in an expression evaluation in greater precision than the result that will be saved.

FLT_EVAL_METH
correctly rounded result

697 If the value being converted is in the range of values that can be represented but cannot be represented exactly, the result is either the nearest higher or nearest lower representable value, chosen in an implementation-defined manner.

Commentary

Although FLT_ROUNDS is only defined to characterize floating-point addition, its value is very likely to characterize the behavior in this case as well.

FLT_ROUNDS

Other Languages

Java specifies the IEEE-754 round-to-nearest mode.

IEC 60559

Common Implementations

Many implementations round-to-nearest. However, rounding to zero (ignoring the least significant bits) can have performance advantages. Changing the rounding mode of the hardware floating-point unit requires an instruction to be executed, which in itself is a fast instruction. However, but a common side effect is to require that the floating-point pipeline be flushed before another floating-point instruction can be issued— an expensive operation. Leaving the processor in round-to-zero mode may offer savings. The IBM S/360, when using its hexadecimal floating point representation, truncates a value when converting it to a representation having less precision.

FLT_ROUNDS

Coding Guidelines

Like other operations on floating values, conversion can introduce a degree of uncertainty into the result. This is a characteristic of floating point; there are no specific guidelines for this situation.

698 If the value being converted is outside the range of values that can be represented, the behavior is undefined.

floating value converted not representable

Commentary

For very small values there is always a higher and lower value that bound them. The situation describes in the C sentence can only occur if the type being demoted from is capable of representing a greater range of exponent values than the destination type.

Other Languages

In Java *A value too small to be represented as a **`float`** is converted to positive or negative zero; a value too large to be represented as a **`float`** is converted to a (positive or negative) infinity.*

Common Implementations

FLT_ROUNDS On many processors the result of the conversion is controlled by the rounding mode. A common behavior is to return the largest representable value (with the appropriate sign) if rounding to zero is in force, and to infinity (with the appropriate sign) if rounding to nearest is in force. When rounding to positive or negative infinity, the result is either the largest representable value or infinity, depending on the sign of the result and the sign of the infinity being rounded to. Many processors also have the ability to raise some form of overflow exception in this situation, which can often be masked.

In signal processing applications infinity is not a sensible value to round to, and processors used for these kinds of applications often saturate at the largest representable value.

Coding Guidelines

The issue of a value not being representable, in the destination type, applies to many floating-point operations other than conversion.

A result of infinity is the required behavior in some applications because it has known properties and its' effect on subsequent calculations might be intended to produce a result of infinity, or zero (for divide operations). Tests can be inserted into the source code to check for infinite results. In other applications, typically graphics or signal processing-oriented ones, a saturated value is required and developers would not expect to need to check for overflow.

Performance is often an important issue in code performing floating-point operations (adding code to do range checks can cause excessive performance penalties). Given the performance issue and the variety of possible application-desired behaviors and actual processor behaviors, there is no obvious guideline recommendation that can be made.

References