Document Number: P0303R0
Date: 2017-10-15
Based on: P0192R1
Authors:

Boris Fomitchev	`boris@stlport.com`
Sergei Nikolaev	`me@cvmlib.com`
Olivier Giroux	`ogiroux@nvidia.com`
Lawrence Crowl	`Lawrence@Crowl.org`

Extensions to C++ for Short Float Type

Contents

List of Tables

1 Scope [intro.scope]

1 This document describes extensions to the C++ Programming Language (1.2) that enable the specification and checking of constraints on template arguments, and the ability to overload functions and specialize class templates based on those constraints. These extensions include new syntactic forms and modifications to existing language semantics.

2 The following referenced document is indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

N4687, 2017-07-30, *Programming Languages — C++* is hereafter called the *C++ Standard*. The numbering of Clauses, sections, and para- graphs in this document reflects the numbering in the C++ Standard. References to Clauses and sections not appearing in this Technical Specification refer to the original, unmodified text in the C++ Standard.

5 Lexical conventions [lex]

5.13.4 Floating literals [lex.fcon]

In this section, ’SF’ and ’sf’ literals shall be added to the floating-suffix list:

floating-suffix: one of:
f l sf F L SF

Also, in paragraph 1, explanation text below the list shall be extended to cover new literal suffixes for short float type:

1 A floating literal consists of an optional prefix specifying a base, an integer part, a radix point, a fraction part, an e, E, p or P, an optionally signed integer exponent, and an optional type suffix.

...

The type of a floating literal is double unless explicitly specified by a suffix. The suffixes f and F specify float, the suffixes sf and SF specify short float, the suffixes l and L specify long double. If the scaled value is not in the range of representable values for its type, the program is ill-formed.

6 Basic concepts [basic]

6.9.1 Fundamental types [basic.fundamental]

Paragraph 8 of section 6.9.1 shall be modified to add short float to the list of floating-point types, and to define its precision in the same manner as it does for the rest of floating-point types:

8 There are ~~three~~ four floating-point types: short float, float, double, and long double. The type double provides at least as much precision as float, the type float provides at least as much precision as short 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 short float is a subset of the set of values of the type float; the set of values of the type double is a subset of the set of values of the type long double. The value representation of floating-point types is implementation-defined.

7 Standard conversions [conv]

7.7 Floating-point promotion [conv.fpprom]

Paragraph 2 for **short float** to float promotion shall be added:

1 A prvalue of type float can be converted to a prvalue of type double. The value is unchanged.

2 A prvalue of type short float can be converted to a prvalue of type double. The value is unchanged.

3 ~~This conversion is~~ Those conversions are called floating-point promotions.

8 Expressions [expr]

Paragraph 11 shall be extended to include a clause for **short float** as follows:

11 Many binary operators that expect operands of arithmetic or enumeration type cause conversions and yield result types in a similar way. The purpose is to yield a common type, which is also the type of the result. This pattern is called the usual arithmetic conversions, which are defined as follows:

— If either operand is of scoped enumeration type (10.2), no conversions are performed; if the other operand does not have the same type, the expression is ill-formed.

— If either operand is of type long double, the other shall be converted to long double. Otherwise, if either operand is double, the other shall be converted to double.

— Otherwise, if either operand is float, the other shall be converted to float.

— Otherwise, if either operand is short float, the other shall be converted to short float.

— Otherwise, the integral promotions (7.6) shall be performed on both operands.

10 Declarations [dcl.dcl]

10.1.7.2 Simple type specifiers [dcl.type.simple]

Table 11 — *simple-type-specifiers* and the types they specify, shall be extended to include a clause for short float as follows:

Table 11 — simple-type-specifiers and the types they specify

Specifier(s)	**Type**
short float	"short float"

11 Declarators [dcl.decl]

11.6.4 List-initialization [dcl.init.list]

In Paragraph 7.2 ("narrowing conversion"), the language shall be changed:

7.2 from ~~long double to double or float , or from double to float,~~ higher precision floating-point value to lower precision one, except where the source is a constant expression and the actual value after conversion is within the range of values that can be represented ...

21 Language support library [language.support]

21.3 Implementation properties [support.limits]

21.3.2 Header <limits> synopsis [limits.syn]

Specialization for short float shall be added:

```
template<> class numeric_limits<short float>;
template<> class numeric_limits<float>;
```

29 Numerics library [numerics]

29.5 Complex numbers [complex.numbers]

The paragraph 2 shall be extended to include short float specializatons.

2 The effect of instantiating the template complex for any type other than short float, float, double, or long double is unspecified. The specializations complex<short float>, complex<float>, complex<double>, and complex<long double> are literal types (6.9).

29.5.1 Header <complex> synopsis [complex.syn]

Specialization for short float shall be added:

```
template<> class complex<short float>;

template<> class complex<float>;
```

...

29.5.3 `complex` specializations [complex.special]

```
  namespace std {
    template<> class complex<short float> {
    public:

      using value_type = short float;

      constexpr complex(short float re = 0.0sf, short float im = 0.0sf);
      constexpr explicit complex(const complex<float>&);
      constexpr explicit complex(const complex<double>&);
      constexpr explicit complex(const complex<long double>&);

      constexpr short float real() const;
      void real(short float);
      constexpr short float imag() const;
      void imag(short float);

      complex<short float>& operator= (short float);
      complex<short float>& operator+=(short float);
      complex<short float>& operator-=(short float);
      complex<short float>& operator*=(short float);
      complex<short float>& operator/=(short float);

      complex<short float>& operator=(const complex<short float>&);
      template<class X> complex<short float>& operator= (const complex<X>&);
      template<class X> complex<short float>& operator+=(const complex<X>&);
      template<class X> complex<short float>& operator-=(const complex<X>&);
      template<class X> complex<short float>& operator*=(const complex<X>&);
      template<class X> complex<short float>& operator/=(const complex<X>&);
  };
```

...

// 29.5.10, complex literals

```
inline namespace literals {
  inline namespace complex_literals {
    constexpr complex<long double> operator""il(long double);
    constexpr complex<long double> operator""il(unsigned long long);
    constexpr complex<double> operator""i(long double);
    constexpr complex<double> operator""i(unsigned long long);
    constexpr complex<float> operator""if(long double);
    constexpr complex<float> operator""if(unsigned long long);
    constexpr complex<short float> operator""isf(long double);
    constexpr complex<short float> operator""isf(unsigned long long);
  }
}
```

...

29.5.10 Suffixes for complex number literals [complex.literals]

1 This section describes literal suffixes for constructing complex number literals. The suffixes i, il, if and ifs create complex numbers of the types complex<double>, complex<long double>, complex<float> and complex<short float> respectively, with their imaginary part denoted by the given literal number and the real part being zero.

...

```
constexpr complex<short float> operator""isf(long double d);
constexpr complex<short float> operator""isf(unsigned long long d);
```

5 Returns: `complex<short float>{0.0sf, static_cast<short float>(d)}.`

29.9 Mathematical functions for floating-point types [c.math]

29.9.1 Header <cmath> synopsis [cmath.syn]

Specialization for short float shall be added:

```
namespace std {
  using short_float_t = see below;
  using float_t = see below;
  using double_t = see below;
```

...

Overloads for short float shall be added:

// 29.9.4, classification / comparison functions

```
int fpclassify(short float x);
bool isfinite(short float x);
bool isinf(short float x);
bool isnan(short float x);
bool isnormal(short float x);
bool signbit(short float x);
bool isgreater(short float x, short float y);
bool isgreaterequal(short float x, short float y);
bool isless(short float x, short float y);
bool islessequal(short float x, short float y);
bool islessgreater(short float x, short float y);
bool isunordered(short float x, short float y);
int fpclassify(short float x);
bool isfinite(short float x);
bool isinf(short float x);
bool isnan(short float x);
bool isnormal(short float x);
bool signbit(short float x);
bool isgreater(short float x, short float y);
bool isgreaterequal(short float x, short float y);
bool isless(short float x, short float y);
bool islessequal(short float x, short float y);
bool islessgreater(short float x, short float y);
bool isunordered(short float x, short float y);}
```

29.9.4 Classification / comparison functions [c.math.fpclass]

1 The classification / comparison functions behave the same as the C macros with the corresponding names defined in the C standard library. Each function is overloaded for the ~~three~~ four floating-point types.

29.9.5 Mathematical special functions [sf.cmath]

3 Overloaded versions for short float are not provided out of precision considerations. If any argument value to any of the functions specified in this subclause are of short float type, standard rules for argument conversion apply.