Document Number:	P0192R0
Date:	2015-11-11
Project:	Programming Language C++, C
Revises:	none
Reply to:	Boris Fomichev boris@stlport.com, Sergei Nikolaev me@cvmlib.com, Olivier Giroux ogiroux@nvidia.com

Adding Fundamental Type for Short Float

1 Objective

- Language support for new shorter float math
- Lexical normalization of floating point types family

2 Reasoning

Why do we need another floating point type in the language? Recently, 16-bit float arithmetic has become very popular in Machine Learning and Image Processing industrial applications. Efficient support for it becomes mission critical for major software products. This year, Adobe's Lightroom uses new HDR DNG format assembling your multiple exposures into a 16-bit *floating point* RAW. This gives those 16-bits much more dynamic range than a traditional file stored as 16 or 32-bit integer data.

The concept of 16-bit arithmetic, in fact, is not new at all. Its complete, functionally mature and standardized. In IEEE 754-2008, the "16-bit base 2" format is officially referred to as `binary16`. Modern ARM CPUs offer native hardware support for (slight variations of) it. There are definite plans to support 16-bit float math natively in upcoming Intel CPUs. Here's an article from Intel mentioning Intel CPUs already have instructions for the 16↔32 bit float conversion (e.g. theres a `_mm256_cvtps_ph` intrinsic to convert from IEEE 32-bit float to binary16).

For GPU hardware, OpenGL provides Small Float Formats for quite a while:

- `GL_HALF_FLOAT` (16 bit, since OpenGL 3.0),
- `GL_HALF_FLOAT_OES` (OpenGL ES 2.0).

OpenGL also has 11 and 10 bit float channels in `GL_R11F_G11F_B10F` and 14-bit in `GL_RGB9_E5`.

Since C++ Standard does not provide a distinct fundamental type to represent floating point values of 16-bit and shorter formats, number of inferior, non-standard solutions is rapidly growing.

The OpenEXR software distribution includes Half, a C++ class for manipulating half float values almost as if they were a built-in C++ data type.

16-bit float matrix math is major new feature of Nvidias CUDA 7.5 platform. Due to lack of C/C++ standard support, `cuda_fp16.h` defines the `half` and `half2` data types as 16-bit C structs and `__half2float()` and `__float2half()` functions for conversion to and from FP32 types, respectively.

GCC introduced `__fp16` native data type extension to support both IEEE and ARM alternative formats of 16-bit float type: *The* `__fp16` *type is a storage format only. For purposes of arithmetic and other operations,* `__fp16` *values in C or C++ expressions are automatically promoted to float. In addition, you cannot declare a function with a return value or parameters of type* `__fp16`.

LLVM has "half" (16-bit floating point value) as first class citizen already. Its time to acknowledge the rise of 16-bit float arithmetic and provide adequate language support for the developers and compiler writers.

3 Proposal

We hope we managed to convince you that C++ is in urgent need for new floating point type. Now, how should we call the new type? First rule of type design is do what the ints do.

So we suggest: **short float**.

This name is intuitive, via `short int` analogy. Also, Common Lisp had it almost exactly like that. Will `short float` play well with the variety of existing platforms, some with shorter float math implementations, and some with none? We bet it will, since for platforms not supporting float types smaller than 32 bits natively, its safe to keep existing practice to fall back to native 32-bit IEEE arithmetic and/or 32-bit storage.

Now lets look at the lexical representation of floating types family names. Introduction of short float just made its lexical irregularity more obvious. What we mean is: while each of the floating types (sort of) acts as at int semantically, some floating point family **type names** do not look similar to the names of their cousins in integer's family at all:

`short int`	`int`	`long int`	`long long int`
`short float`	`float`	`double`	`long double`

We propose adding the following new type and type aliases to the Standard:

`short float`	new float type, shorter than 32 bit
`long float`	alias for `double`, 64 bit
`long long float`	alias for `long double`, potentially 128 bit

We also propose adding literal suffixes: **s** and **S**, to specify floating literals of short float type, i.e. `1.23s` or `1.23e-1S`.

4 Implementation

As of storage and bit-layout for a short float number, we would expect most implementations to follow IEEE 754-2008 half-precision floating point number format. Since its not guaranteed, `numeric_limits<short float>` methods should be used to check IEEE conformance. Short float arithmetic: we suggest native support on platforms where `binary16` is supported in hardware (ARM64, CUDA etc). On platforms that lack native 16-bit float support, existing way of handling `binary16` operands should be preserved: conversion to 32-bit float on read from memory → float math operation → conversion back to 16 bit for memory store. In CUDA 7.5 platform, `cuda_fp16.h` defines the `half` and `half2` datatypes and `__half2float()` and `__float2half()` functions for conversion to and from FP32 types, respectively. The 3rd generation Intel Core processor family already introduced two half-float conversion instructions: `vcvtps2ph` for converting from 32-bit float to half-float, and `vcvtph2ps` for converting from half-float to 32-bit float.

5 New cstdfloat header

Similar to `<cstdint>` we propose adding new header `<cstdfloat>` with the following type definitions:

```
// architecture dependent, might be 32 bit
typedef short float      float16_t;

typedef float            float32_t;

typedef long float       float64_t;

// architecture dependent,
// might be 64 bit
typedef long long float  float128_t;
```

As per the implementation given above we also set

```
// Smallest positive short float
#define SFLT_MIN         5.96046448e-08

// Smallest positive
// normalized short float
#define SFLT_NRM_MIN     6.10351562e-05

// Largest positive short float
#define SFLT_MAX         65504.0

// Smallest positive e
// for which (1.0 + e) != (1.0)
#define SFLT_EPSILON     0.00097656

// Number of digits in mantissa
// (significand + hidden leading 1)
#define SFLT_MANT_DIG    11

// Number of base 10 digits that
// can be represented without change
```

```
#define SFLT_DIG        2

// Base of the exponent
#define SFLT_RADIX      2

// Minimum negative integer such that
// HALF_RADIX raised to the power of
// one less than that integer is a
// normalized short float
#define SFLT_MIN_EXP    -13

// Maximum positive integer such that
// HALF_RADIX raised to the power of
// one less than that integer is a
// normalized short float
#define SFLT_MAX_EXP    16

// Minimum positive integer such
// that 10 raised to that power is
// a normalized short float
#define SFLT_MIN_10_EXP -4

// Maximum positive integer such
// that 10 raised to that power is
// a normalized short float
#define SFLT_MAX_10_EXP  4
```

6 Conversions

Conversions from short float to float are lossless; all short float numbers are exactly representable as floats.

Conversions from float to short float may not preserve the float's value exactly. If a float is not representable as a short float, the float value is rounded to the nearest representable half using one of the following `std::float_round_style` modes:

```
std::round_indeterminate
std::round_toward_zero
std::round_to_nearest
std::round_toward_infinity
std::round_toward_neg_infinity
```

Overflows during float-to-short float conversions cause arithmetic exceptions. An overflow occurs when the float value to be converted is too large to be represented as a short float, or if the float value is an infinity or a NAN.