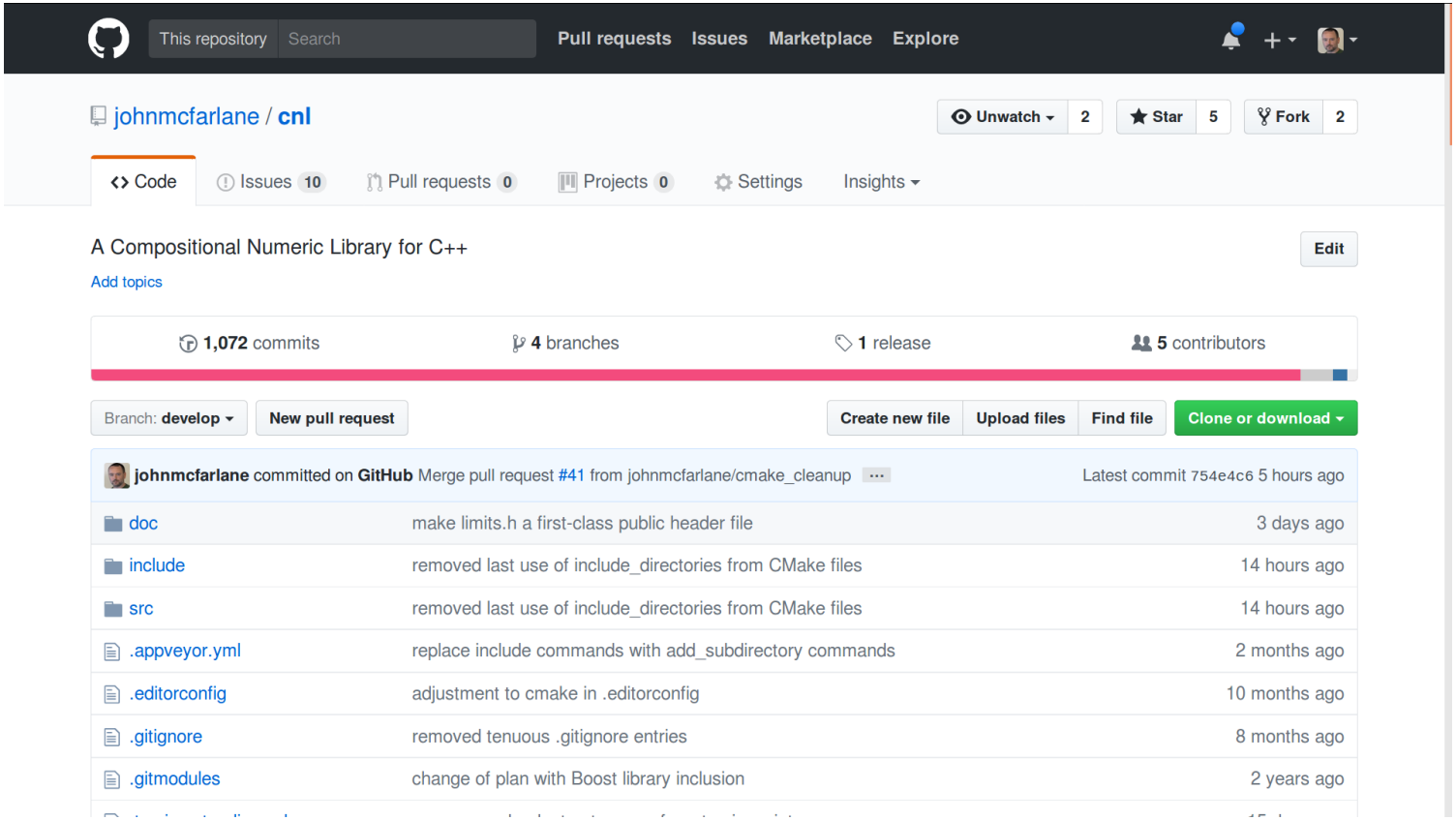
CNL: A Compositional Numeric Library

John McFarlane

A9.com

Background

Background


This repository
Search
Pull requests
Issues
Marketplace
Explore
johnmcfarlane / cnl
Unwatch 2
Star 5
Fork 2
Code
Issues 10
Pull requests 0
Projects 0
Settings
Insights
A Compositional Numeric Library for C++
Edit
Add topics
1,072 commits
4 branches
1 release
5 contributors
Branch: develop
New pull request
Create new file
Upload files
Find file
Clone or download
johnmcfarlane committed on GitHub Merge pull request #41 from johnmcfarlane/cmake_cleanup
Latest commit 754e4c6 5 hours ago
doc
make limits.h a first-class public header file
3 days ago
include
removed last use of include_directories from CMake files
14 hours ago
src
removed last use of include_directories from CMake files
14 hours ago
.appveyor.yml
replace include commands with add_subdirectory commands
2 months ago
.editorconfig
adjustment to cmake in .editorconfig
10 months ago
.gitignore
removed tenuous .gitignore entries
8 months ago
.gitmodules
change of plan with Boost library inclusion
2 years ago

The Problem with Integers

The Problem with Integers

1. Low resolution

The Problem with Integers

1. Low resolution
2. Limited range

The Problem with Integers

1. Low resolution
2. Limited range
3. 'Interesting' behavior of arithmetic operations

The Problem with Floating-Point

1. Complicated
    - ± 1.significand * 2^{exponent}
    - special values, denormalized values, -0

The Problem with Floating-Point

1. Complicated
    - ± 1.significand * 2^{exponent}
    - special values, denormalized values, -0
2. Occasional weirdness can surprise:
    - determinism, associativity, commutativity and ordering

The Problem with Floating-Point

1. Complicated
    - ± 1.significand * 2 $^{\text{exponent}}$
    - special values, denormalized values, -0
2. Occasional weirdness can surprise:
    - determinism, associativity, commutativity and ordering
3. `<cmath>` functions lack `constexpr`

The Problem with Floating-Point

1. Complicated
    - ± 1.significand * 2 $^{\text{exponent}}$
    - special values, denormalized values, -0
2. Occasional weirdness can surprise:
    - determinism, associativity, commutativity and ordering
3. `<cmath>` functions lack `constexpr`
4. Variable resolution

The Problem with Floating-Point

1. Complicated
    - ± 1.significand * 2 $^{\text{exponent}}$
    - special values, denormalized values, -0
2. Occasional weirdness can surprise:
    - determinism, associativity, commutativity and ordering
3. `<cmath>` functions lack `constexpr`
4. Variable resolution
5. Costly in energy and silicon

Analysis

Analysis

- Floating-point problems are not *so* bad.

Analysis

- Floating-point problems are not *so* bad.
- Integers are a powerful abstraction over registers.

Analysis

- Floating-point problems are not *so* bad.
- Integers are a powerful abstraction over registers.
- But we can do a lot better.

Analysis

- Floating-point problems are not *so* bad.
- Integers are a powerful abstraction over registers.
- But we can do a lot better.
- so ...

Goal of CNL

"Do for `int` what the STL did for `[]`."

Goal of CNL

"**Do for `int` what the STL did for `[]`.**"

- Provide zero-cost abstractions over language-level features:

```
std::array<T, N> a;  // T a[N]
std::array<T, N>::iterator i = std::begin(a); // T* i
```

Goal of CNL

"**Do for `int` what the STL did for `[]`.**"

- Provide zero-cost abstractions over language-level features:

```
std::array<T, N> a;  // T a[N]
std::array<T, N>::iterator i = std::begin(a); // T* i
```

- Maintain a familiar interface:

```
auto const& third = a[2];
for (auto const& element : a) { /* ... */ }
```

Goal of CNL

"**Do for `int` what the STL did for `[]`.**"

- Provide zero-cost abstractions over language-level features:

```
std::array<T, N> a;  // T a[N]
std::array<T, N>::iterator i = std::begin(a); // T* i
```

- Maintain a familiar interface:

```
auto const& third = a[2];
for (auto const& element : a) { /* ... */ }
```

- Allow users to opt in to positive-cost functionality:

```
std::array<T, N> a{n};
auto const& bad_element = a.at(n+1);  // gotcha!
```

Goal of CNL

"**Do for `int` what the STL did for `[]`.**"

- Provide zero-cost abstractions over language-level features:

```
std::array<T, N> a;  // T a[N]
std::array<T, N>::iterator i = std::begin(a); // T* i
```

- Maintain a familiar interface:

```
auto const& third = a[2];
for (auto const& element : a) { /* ... */ }
```

- Allow users to opt in to positive-cost functionality:

```
std::array<T, N> a{n};
auto const& bad_element = a.at(n+1);  // gotcha!
```

- And most importantly...

Goal of CNL

"**Do for `int` what the STL did for `[]`.**"

- Compose!

```
using fs_cache = unordered_map<filesystem::path, vector<byte>>;
```

Non-Goal

"**Don't do for `int` what STL doesn't do for `[]`.**"

Non-Goal

"**Don't do for `int` what STL doesn't do for `[]`.**"

- Don't make the user pay for what they don't use.

Fixed-Point Arithmetic

Definition:

```
// cnl/fixed_point.h
namespace cnl {

    template<typename Rep = int, int Exponent = 0>
    class fixed_point {
        // ...
    private:
        Rep r;
    };

}
```

Fixed-Point Arithmetic

Definition:

```
// cnl/fixed_point.h
namespace cnl {

    template<typename Rep = int, int Exponent = 0>
    class fixed_point {
        // ...
    private:
        Rep r;
    };

}
```

Example usage:

```
using cnl::fixed_point;

void f() {
    auto n = fixed_point<int, -8>{ 0.25 };
    std::cout << n * 5; // prints "1.25"
}
```

The Good

The Good

```
bool foo(float f) {
    auto fixed = fixed_point<int, -16>{f};
    auto fixed_plus_one = fixed + 1;
    return fixed_plus_one > fixed;
}
```

The Good

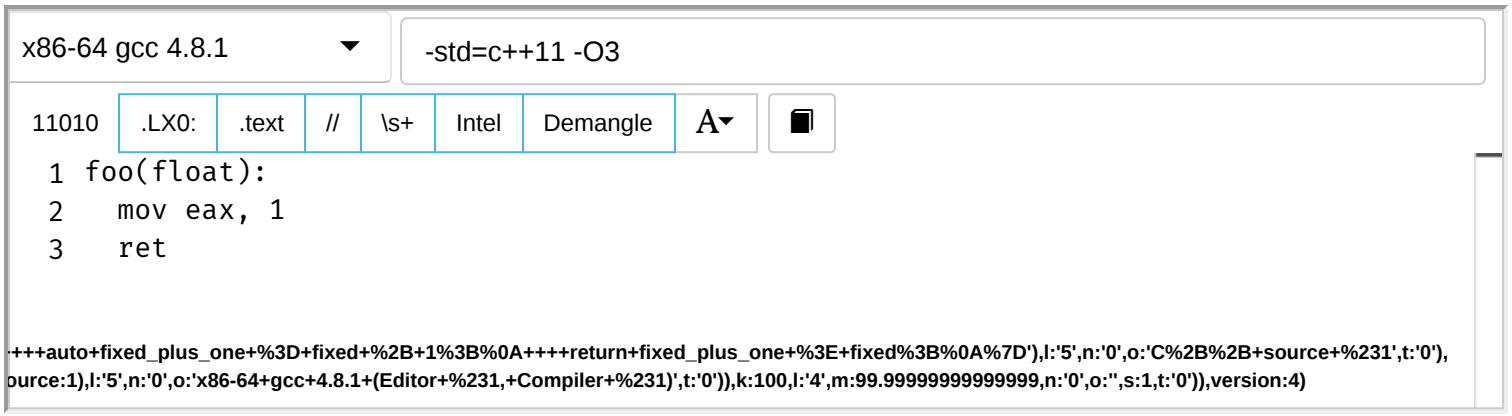
```
bool foo(float f) {
    auto fixed = fixed_point<int, -16>{f};
    auto fixed_plus_one = fixed + 1;
    return fixed_plus_one > fixed;
}
```

```
bool foo(float) {
    return true;
}
```

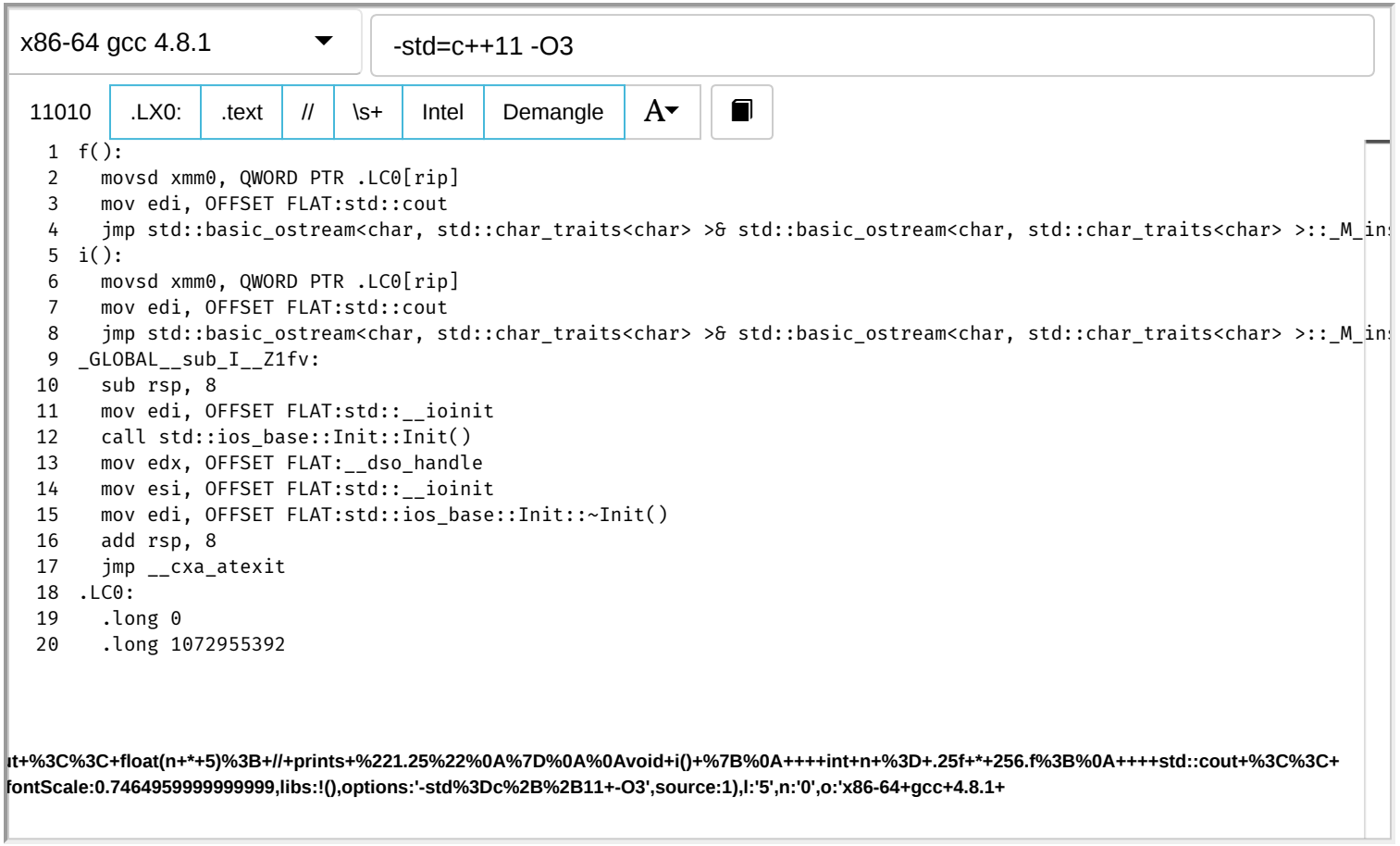
The Good

```
bool foo(float f) {
    auto fixed = fixed_point<int, -16>{f};
    auto fixed_plus_one = fixed + 1;
    return fixed_plus_one > fixed;
}
```

```
bool foo(float) {
    return true;
}
```



The Good


x86-64 gcc 4.8.1
-std=c++11 -O3
11010
.LX0:
.text
//
\s+
Intel
Demangle
f():
movsd xmm0, QWORD PTR .LC0[rip]
mov edi, OFFSET FLAT:std::cout
jmp std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char, std::char_traits<char> >::_M_in
i():
movsd xmm0, QWORD PTR .LC0[rip]
mov edi, OFFSET FLAT:std::cout
jmp std::basic_ostream<char, std::char_traits<char> >& std::basic_ostream<char, std::char_traits<char> >::_M_in
_GLOBAL__sub_I__Z1fv:
sub rsp, 8
mov edi, OFFSET FLAT:std::__ioinit
call std::ios_base::Init::Init()
mov edx, OFFSET FLAT:__dso_handle
mov esi, OFFSET FLAT:std::__ioinit
mov edi, OFFSET FLAT:std::ios_base::Init::~Init()
add rsp, 8
jmp __cxa_atexit
.LC0:
.long 0
.long 1072955392
ıt+%3C%3C+float(n+*+5)%3B+//+prints+%221.25%22%0A%7D%0A%0Avoid+i()+%7B%0A++++int+n+%3D+.25f+*+256.f%3B%0A++++std::cout+%3C%3C+
fontScale:0.7464959999999999,libs:!(),options:'-std%3Dc%2B%2B11+-O3',source:1),l:'5',n:'0',o:'x86-64+gcc+4.8.1+

The Bad

The Bad

```
// undefined behavior
auto a = numeric_limits<fixed_point<int, -16>>::max() + 1;
```

The Bad

```
// undefined behavior
auto a = numeric_limits<fixed_point<int, -16>>::max() + 1;
```

```
// compiles
static_assert(1 == 1, "this does compile");

// error: static assertion failed: this does not compile
static_assert(1 != 1, "this does not compile");

// error: left shift count >= width of type
static_assert(1 << 1000, "this does not compile");
```

The Bad

```
// undefined behavior
auto a = numeric_limits<fixed_point<int, -16>>::max() + 1;
```

```
// compiles
static_assert(1 == 1, "this does compile");

// error: static assertion failed: this does not compile
static_assert(1 != 1, "this does not compile");

// error: left shift count >= width of type
static_assert(1 << 1000, "this does not compile");
```

```
static_assert(numeric_limits<fixed_point<int, -16>>::max() - 1, "OK");
static_assert(numeric_limits<fixed_point<int, -16>>::max() + 1, "error");
```

The Bad

```
static_assert(fixed_point<unsigned>{1} < fixed_point<signed>{-1}, "OK(!)");
```

The Bad

```
static_assert(fixed_point<unsigned>{1} < fixed_point<signed>{-1}, "OK(!)");
```

```
static_assert(numeric_limits<int>::max() + 1, "error");
static_assert(unsigned{1} < signed{-1}, "evaluates to true");
```

The Bad

```
static_assert(fixed_point<unsigned>{1} < fixed_point<signed>{-1}, "OK(!)");
```

```
static_assert(numeric_limits<int>::max() + 1, "error");
static_assert(unsigned{1} < signed{-1}, "evaluates to true");
```

```
static_assert(fixed_point{1u} < fixed_point{-1});
```

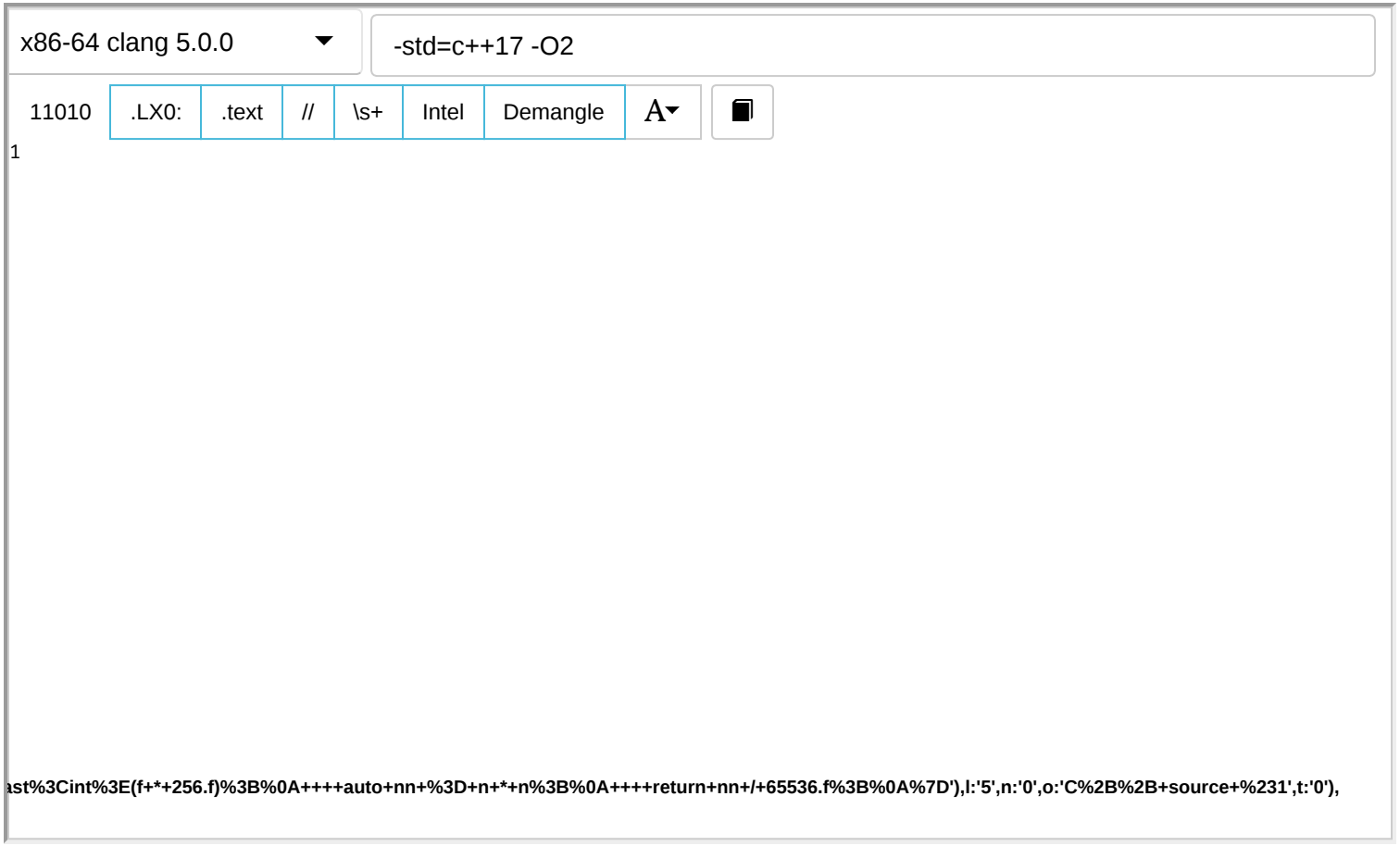
The Ugly

The Ugly

```
auto n = fixed_point<int, -8>{1.5};
auto nn = n * n;    // fixed_point<int, -16>;

static_assert(std::is_same_v<decltype(nn), fixed_point<int, -16>>);
```

The Ugly


x86-64 clang 5.0.0
-std=c++17 -O2
11010
.LX0:
.text
//
\s+
Intel
Demangle
1
ast%3Cint%3E(f+*+256.f)%3B%0A++++auto+nn+%3D+n+*+n%3B%0A++++return+nn+/+65536.f%3B%0A%7D'),l:'5',n:'0',o:'C%2B%2B+source+%231',t:'0'),

The Ugly

The Ugly

```
constexpr auto n = fixed_point<int, -8>{1.5};
constexpr auto d = fixed_point<int, -8>{2.25};
constexpr auto q = n / d;    // fixed_point<int, 0>;
```

The Ugly

```
constexpr auto n = fixed_point<int, -8>{1.5};
constexpr auto d = fixed_point<int, -8>{2.25};
constexpr auto q = n / d;    // fixed_point<int, 0>;
```

```
constexpr auto n = fixed_point<int, -8>{1.5};
constexpr auto d = fixed_point<int, -8>{2.25};
constexpr auto q = cnl::divide(n, d);    // fixed_point<long, -31>;
```

How Do You Solve a Problem Like Division?

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

```
1 / 100 = 0.01
10 / 5.5 = 1.818181818181...
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

```
1 / 100 = 0.01
10 / 5.5 = 1.818181818181...
```

```
template<typename Integer> class fraction { Integer numerator, denominator; ... };
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

```
1 / 100 = 0.01
10 / 5.5 = 1.818181818181...
```

```
template<typename Integer> class fraction { Integer numerator, denominator; ... };
```

```
AAA.BBBBB * CCCCCC.DD = AAACCCCCC.BBBBBDD
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

```
1 / 100 = 0.01
10 / 5.5 = 1.818181818181...
```

```
template<typename Integer> class fraction { Integer numerator, denominator; ... };
```

```
AAA.BBBBB * CCCCCC.DD = AAACCCCCC.BBBBBDD
```

```
AAA.BBBBB / CCCCCC.DD = AAADD.BBBBBCCCCCC
```

How Do You Solve a Problem Like Division?

```
5.5 * 5.5 = 30.25
55. * .55 = 30.25
```

```
1 / 100 = 0.01
10 / 5.5 = 1.818181818181...
```

```
template<typename Integer> class fraction { Integer numerator, denominator; ... };
```

```
AAA.BBBBB * CCCCCC.DD = AAACCCCCC.BBBBBDD
```

```
AAA.BBBBB / CCCCCC.DD = AAADD.BBBBBCCCCCC
```

```
constexpr auto n = fixed_point<int, -8>{1.5};
constexpr auto d = fixed_point<int, -8>{2.25};
constexpr auto q = cnl::divide(n, d);    // fixed_point<long, -31>;
```

Elasticity

Elasticity

```
auto n = fixed_point<uint8_t, -8>{0.99609375};
auto nn = n * n;
```

Elasticity

```
auto n = fixed_point<uint8_t, -8>{0.99609375};
auto nn = n * n;    // fixed_point<int, -16>{0.9922027587890625};
```

Elasticity

```
auto n = fixed_point<uint8_t, -8>{0.99609375};
auto nn = n * n;    // fixed_point<int, -16>{0.9922027587890625};
```

```
auto n = fixed_point<int, -31>{0.99609375};
auto nn = n * n;    // fixed_point<int, -62>{?!?!?!?!?!?!};
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

```
auto ee = e * e;   // elastic_integer<62>{INT64_C(0x3FFFFFFF00000001)}
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

```
auto ee = e * e;   // elastic_integer<62>{INT64_C(0x3FFFFFFF00000001)}
```

```
auto _2ee = ee + ee;   // elastic_integer<63>{INT64_C(0x7FFFFFFE00000002)}
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

```
auto ee = e * e;   // elastic_integer<62>{INT64_C(0x3FFFFFFF00000001)}
```

```
auto _2ee = ee + ee;   // elastic_integer<63>{INT64_C(0x7FFFFFFE00000002)}
```

```
auto fpe = fixed_point<elastic_integer<31>, -31>{0.99609375};
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

```
auto ee = e * e;   // elastic_integer<62>{INT64_C(0x3FFFFFFF00000001)}
```

```
auto _2ee = ee + ee;   // elastic_integer<63>{INT64_C(0x7FFFFFFE00000002)}
```

```
auto fpe = fixed_point<elastic_integer<31>, -31>{0.99609375};
```

```
auto sq = fpe * fpe;  // fixed_point<elastic_integer<62>, -62>{0.9922027587890625}
```

Elasticity

```
template<int Digits, class Narrowest = int>
class elastic_integer { WideEnoughInteger r; /* other stuff */ };
```

```
#include <cnl/elastic_integer.h>
using cnl::elastic_integer;
auto e = elastic_integer<31>{0x7FFFFFFF};   // r has 31 or more digits
```

```
auto ee = e * e;   // elastic_integer<62>{INT64_C(0x3FFFFFFF00000001)}
```

```
auto _2ee = ee + ee;   // elastic_integer<63>{INT64_C(0x7FFFFFFE00000002)}
```

```
auto fpe = fixed_point<elastic_integer<31>, -31>{0.99609375};
```

```
auto sq = fpe * fpe;  // fixed_point<elastic_integer<62>, -62>{0.9922027587890625}
```

```
#include <cnl/auxiliary/elastic_fixed_point.h>
auto q = sq / sq; // fixed_point<elastic_integer<124>, -62>{1}, q), "");
```

Run-time Safety

Run-time Safety

```
#include <cnl/safe_integer.h>
using cnl::safe_integer;
auto i = safe_integer<uint8_t>{255};
```

Run-time Safety

```
#include <cnl/safe_integer.h>
using cnl::safe_integer;
auto i = safe_integer<uint8_t>{255};
```

```
auto j = i + 1;
```

Run-time Safety

```
#include <cnl/safe_integer.h>
using cnl::safe_integer;
auto i = safe_integer<uint8_t>{255};
```

```
auto j = i + 1; // safe_integer<int>{256}
```

Run-time Safety

```
#include <cnl/safe_integer.h>
using cnl::safe_integer;
auto i = safe_integer<uint8_t>{255};
```

```
auto j = i + 1; // safe_integer<int>{256}
```

```
safe_integer<uint8_t> k = i + 1;  // throw std::overflow_error
```

Run-time Safety

```
#include <cnl/safe_integer.h>
using cnl::safe_integer;
auto i = safe_integer<uint8_t>{255};
```

```
auto j = i + 1; // safe_integer<int>{256}
```

```
safe_integer<uint8_t> k = i + 1;  // throw std::overflow_error
```

```
constexpr safe_integer<uint8_t> k = i + 1;
static_assert(cnl::_impl::identical(safe_integer<int>{256}, k));
```

```
[ 29%] Building CXX object src/test/CMakeFiles/fp_test.dir/cppcon2017.cpp.o
/home/john/cnl/src/test/cppcon2017.cpp:151:37: fatal error: constexpr variable 'k' must be i
    constexpr safe_integer<uint8_t> k = i + 1;
                                    ^~~~~~~~~
/home/john/cnl/include/cnl/overflow.h:52:40: note: subexpression not valid in a constant exp
            return condition ? value : throw std::overflow_error("");
                                       ^
```

and so on...

Deduction and UDLs

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

```
auto y = fixed_point{128};  // fixed_point<int, 0>{1}
```

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

```
auto y = fixed_point{128};  // fixed_point<int, 0>{1}
```

```
using cnl::literals;
auto z = fixed_point{128_c};  // fixed_point<int, 7>{128}
```

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

```
auto y = fixed_point{128};  // fixed_point<int, 0>{1}
```

```
using cnl::literals;
auto z = fixed_point{128_c};  // fixed_point<int, 7>{128}
```

```
auto a = fixed_point{0b10000000000000000000000000000000000000000_c};
// a === fixed_point<int, 40>{0b10000000000000000000000000000000000000000l}
```

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

```
auto y = fixed_point{128};  // fixed_point<int, 0>{1}
```

```
using cnl::literals;
auto z = fixed_point{128_c};  // fixed_point<int, 7>{128}
```

```
auto a = fixed_point{0b10000000000000000000000000000000_c};
// a === fixed_point<int, 40>{0b1000000000000000000000000000000000000000l}
```

```
auto b = fixed_point{0b11111111111111111111111111111111_c};
// b === fixed_point<long, 0>{0b11111111111111111111111111111111l}
```

Deduction and UDLs

```
auto x = fixed_point{42ul}; // fixed_point<unsigned long, 0>{42}
```

```
auto y = fixed_point{128};  // fixed_point<int, 0>{1}
```

```
using cnl::literals;
auto z = fixed_point{128_c};  // fixed_point<int, 7>{128}
```

```
auto a = fixed_point{0b10000000000000000000000000000000000000000_c};
// a === fixed_point<int, 40>{0b10000000000000000000000000000000000000000l}
```

```
auto b = fixed_point{0b11111111111111111111111111111111111111111_c};
// b === fixed_point<long, 0>{0b11111111111111111111111111111111111111111l}
```

Deduction and UDLs

Deduction and UDLs

```
auto c = elastic_integer{2017_c}; // elastic_integer<11>{2017}
```

Deduction and UDLs

```
auto c = elastic_integer{2017_c}; // elastic_integer<11>{2017}
```

```
auto e = 0x7f000_elastic; // fixed_point<elastic_integer<7>, 12>{0x7f000}
```

Interoperability - Boost.Multiprecision

```
#include <cnl/auxiliary/boost.multiprecision.h>
using namespace boost::multiprecision;

template<int NumBits, int Exponent = 0>
using mp_fixed_point = cnl::fixed_point<
        number<cpp_int_backend<NumBits, NumBits, signed_magnitude, unchecked, void
        Exponent>;
```

Interoperability - Boost.Multiprecision

```
#include <cnl/auxiliary/boost.multiprecision.h>
using namespace boost::multiprecision;

template<int NumBits, int Exponent = 0>
using mp_fixed_point = cnl::fixed_point<
        number<cpp_int_backend<NumBits, NumBits, signed_magnitude, unchecked, void
        Exponent>;
```

Fixed-Point + Boost.Multiprecision:

Interoperability - Boost.Multiprecision

```
#include <cnl/auxiliary/boost.multiprecision.h>
using namespace boost::multiprecision;

template<int NumBits, int Exponent = 0>
using mp_fixed_point = cnl::fixed_point<
        number<cpp_int_backend<NumBits, NumBits, signed_magnitude, unchecked, void
        Exponent>;
```

Fixed-Point + Boost.Multiprecision:

- googol (10^{100}) ✓

Interoperability - Boost.Multiprecision

```
#include <cnl/auxiliary/boost.multiprecision.h>
using namespace boost::multiprecision;

template<int NumBits, int Exponent = 0>
using mp_fixed_point = cnl::fixed_point<
        number<cpp_int_backend<NumBits, NumBits, signed_magnitude, unchecked, void
        Exponent>;
```

Fixed-Point + Boost.Multiprecision:

- googol (10^{100}) ✓
- googolth (1 / googol) ✓

Interoperability - Boost.Multiprecision

```
#include <cnl/auxiliary/boost.multiprecision.h>
using namespace boost::multiprecision;

template<int NumBits, int Exponent = 0>
using mp_fixed_point = cnl::fixed_point<
        number<cpp_int_backend<NumBits, NumBits, signed_magnitude, unchecked, void
        Exponent>;
```

Fixed-Point + Boost.Multiprecision:

- googol (10^{100}) ✓
- googolth (1 / googol) ✓
- googolplex (10^{googol}) ✗

Interoperability - Boost.SIMD

```
#include <cnl/auxiliary/boost.simd.h>
using boost::simd::pack;

template<class T, std::size_t N, int Exponent>
using fixed_point_pack = fixed_point<pack<T, N>, Exponent>;
```

Interoperability - Boost.SIMD

```
#include <cnl/auxiliary/boost.simd.h>
using boost::simd::pack;

template<class T, std::size_t N, int Exponent>
using fixed_point_pack = fixed_point<pack<T, N>, Exponent>;
```

```
using fpp = fixed_point_pack<int, 4, -16>;
using initializer = initializer<fpp>;

auto expected = fpp{initializer{7.9375+-1, -8.+.125, 0+-5, 3.5+-3.5}};
auto augend = fpp{initializer{7.9375, -8., 0, 3.5}};
auto addend = fpp{initializer{-1, .125, -5, -3.5}};
auto sum = augend + addend;
```

Interoperability - Boost.SIMD

```
#include <cnl/auxiliary/boost.simd.h>
using boost::simd::pack;

template<class T, std::size_t N, int Exponent>
using fixed_point_pack = fixed_point<pack<T, N>, Exponent>;
```

```
using fpp = fixed_point_pack<int, 4, -16>;
using initializer = initializer<fpp>;

auto expected = fpp{initializer{7.9375+-1, -8.+.125, 0+-5, 3.5+-3.5}};
auto augend = fpp{initializer{7.9375, -8., 0, 3.5}};
auto addend = fpp{initializer{-1, .125, -5, -3.5}};
auto sum = augend + addend;
```

CNL Today and Tomorrow

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

- Arbitrary width

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

- Arbitrary width
- Full Support for Rounding and Overflow

```
template<class Rep = int, class RoundingTag = closest_rounding_tag>
class precise_integer;
```

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

- Arbitrary width
- Full Support for Rounding and Overflow

```
template<class Rep = int, class RoundingTag = closest_rounding_tag>
class precise_integer;
```

- Full complement of operators for `safe_integer` and `precise_integer`

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

- Arbitrary width
- Full Support for Rounding and Overflow

```
template<class Rep = int, class RoundingTag = closest_rounding_tag>
class precise_integer;
```

- Full complement of operators for `safe_integer` and `precise_integer`
- Full complement of free functions

```
add(saturated_overflow, UINT32_C(0xFFFFFFFF), UINT32_C(0x12345678))
divide(closest_rounding_tag, 2, 3);
```

CNL Today and Tomorrow

github.com/johnmcfarlane/cnl

- Arbitrary width
- Full Support for Rounding and Overflow

```
template<class Rep = int, class RoundingTag = closest_rounding_tag>
class precise_integer;
```

- Full complement of operators for `safe_integer` and `precise_integer`
- Full complement of free functions

```
add(saturated_overflow, UINT32_C(0xFFFFFFFF), UINT32_C(0x12345678))
divide(closest_rounding_tag, 2, 3);
```

- Better Literals

```
auto a = 0b1111.1111_elastic;  // fixed_point<elastic_integer<8>, -4>
```