Engineering Software

Integral types

Andrei Zlate-Podani

1968 NATO Software Engineering Conference - Garmisch

- Projects running over-budget
- Projects running over-time
- Software was inefficient
- Software was of low quality
- Software often did not meet requirements
- Projects were unmanageable and code difficult to maintain
- Software was never delivered

Writing software is bottom – up

- Larger constructs are built by using basic operations and / or calling functions.
- To preserve correctness it is necessary, but not sufficient, to satisfy the preconditions of the basic operations and functions.
- Any errors need to be detected and reported to the next layer, unless they are dealt with locally.

Contracts

```
template<class _InputIt, class _Pred>
// Requires InputIterator, Predicate
_InputIt
find_if_not(_InputIt first, _InputIt last, _Pred pred)
{
    for (; first != last; ++first)
        if (!pred(*first))
            break;
    return first;
}
```

Contracts – find_if_not

- No conversion from iterator’s value type to predicate’s parameter type.
  - Assume a range of **`float`** and a predicate that takes **`int`**
- The values in the range must be independent of the adjoining ones
  - Assume a range over an UTF-8 string

- The meaning associated with the values must be the same in the range and in the predicate.
  - Assume that the predicate is looking values in the metric system and the range uses imperial measures.

bool

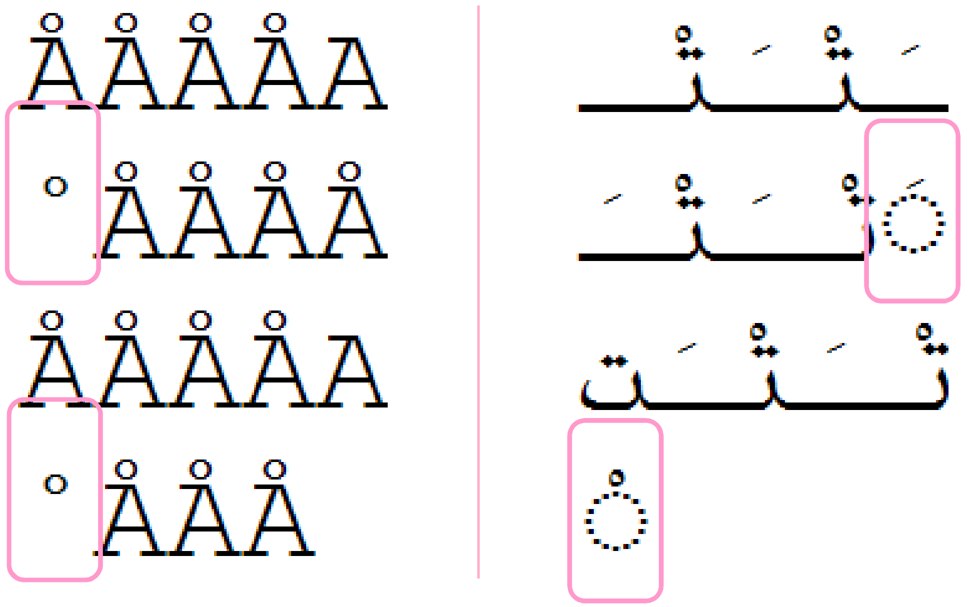
- `false – true + true == false`
- `false XOR true OR true == true`

Characters

- `char` distinct from `signed char` and `unsigned char`
- `wchar_t` distinct type, sign and size are implementation defined
- `char16_t` and `char32_t` are **not** fixed size
- Numerical values who’s meaning is given by the encoding
- The Unicode standard defines an N-to-1 relationship between code-points and glyphs

Å

- **U+00C5** (latin capital letter a with ring above)
- **U+212B** (ångström symbol)
- **U+0041 U+030A** ('A' + combining ring above)



boost
C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

```
// &#...; - assumes ASCII
case Ch('#'):
    if (src[2] == Ch('x'))
    {                           // &#xHHHH -> maximum value 10FFFF
        unsigned long code = 0;
        src += 3;   // Skip &#x
        while (1)
        {
            uint8_t digit =
                g_digits_tab[static_cast<uint8_t>(*src)];

            if (digit == 0xFF) break;

            code = code * 16 + digit;
            ++src
        }

        insert_coded_character<Flags>(dest, code);
    }
```

avast

boost
C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

```
// &#...; - assumes ASCII
case Ch('#'):
    if (src[2] == Ch('x'))
    {                        // &#xHHHH -> maximum value 10FFFF
        unsigned long code = 0;
        src += 3;   // Skip &#x
        while (1)
        {
            uint8_t digit =
                g_digits_tab[static_cast<uint8_t>(*src)];

            if (digit == 0xFF) break;

            code = code * 16 + digit;
            ++src
        }

        insert_coded_character<Flags>(dest, code);
    }
```

avast

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

```
// &#...; - assumes ASCII
case Ch('#'):
    if (src[2] == Ch('x'))
    {                       // &#xHHHH -> maximum value 10FFFF
        unsigned long code = 0;
        src += 3;   // Skip &#x
        while (1)
        {
            uint8_t digit =
                g_digits_tab[static_cast<uint8_t>(*src)];

            if (digit == 0xFF) break;

            code = code * 16 + digit;
            ++src
        }

        insert_coded_character<Flags>(dest, code);
    }
```

boost C++ LIBRARIES

"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

```
// &#...; - assumes ASCII
case Ch('#'):
    if (src[2] == Ch('x'))
    {                           // &#xHHHH -> maximum value 10FFFF
        unsigned long code = 0;
        src += 3;   // Skip &#x
        while (1)
        {
            uint8_t digit =
                g_digits_tab[static_cast<uint8_t>(*src)];

            if (digit == 0xFF) break;

            code = code * 16 + digit;
            ++src
        }

        insert_coded_character<Flags>(dest, code);
    }
```

avast

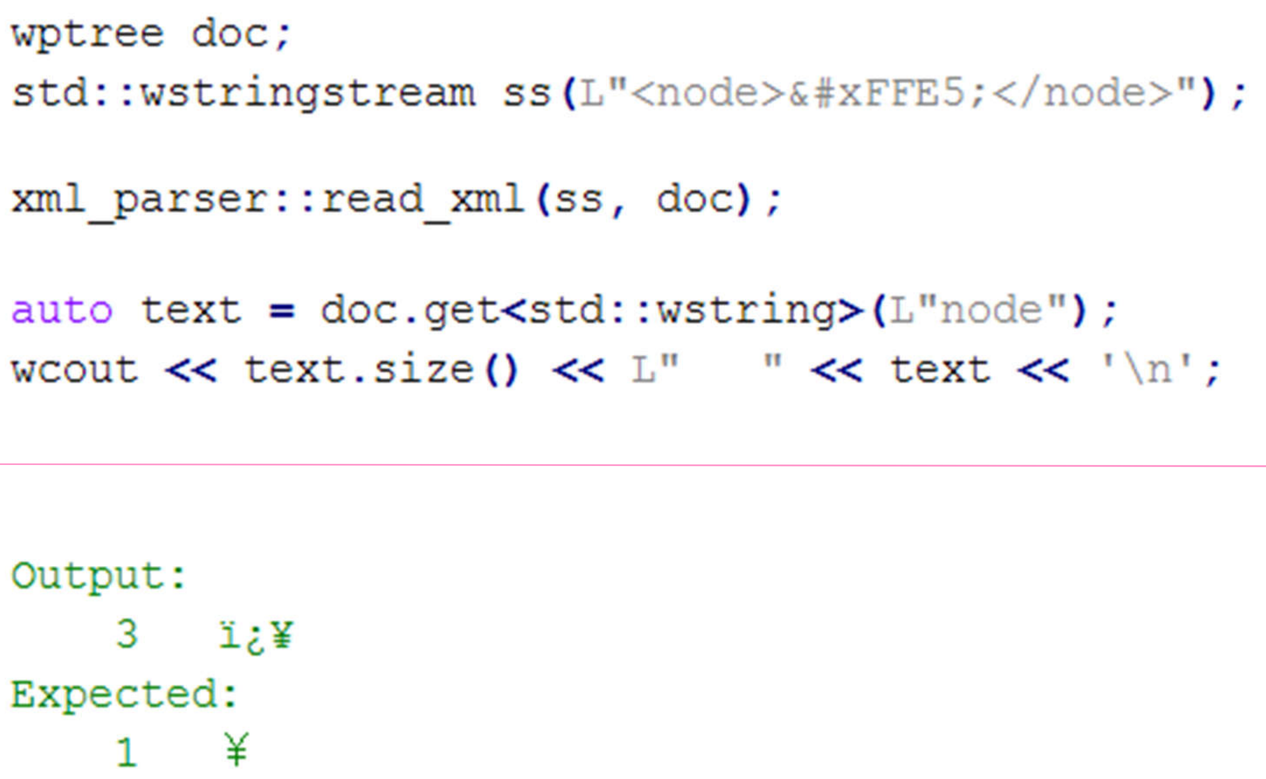
"...one of the most highly regarded and expertly designed C++ library projects in the world."
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

```
// &#...; - assumes ASCII
case Ch('#'):
                                                            FFF
```

```
// Insert coded character, using UTF8 or 8-bit ASCII
template<int Flags>
void insert_coded_character(Ch *&text, ulong code)
{
    if (Flags & parse_no_utf8)
    {
        // Insert 8-bit ASCII character
    }
    else
    {
       // Insert UTF8 sequence
```

```
        }
        insert_coded_character<Flags>(dest, code);
    }
```

Broken



```
wptree doc;
std::wstringstream ss(L"<node>&#xFFE5;</node>");

xml_parser::read_xml(ss, doc);

auto text = doc.get<std::wstring>(L"node");
wcout << text.size() << L"    " << text << '\n';
```

```
Output:
    3    ï¿¥
Expected:
    1    ￥
```

Properties for signed integers

- Addition is associative – partially
- Addition is commutative – partially for sequences
- Multiplication is associative & commutative – yes
- Multiplication is distributive – partially
- Division is distributive (`(a + b) / c`) – no
- Division is the inverse of multiplication – partially
- Multiplication if the inverse of division – no

Integral promotions

```
uint8_t a = 127, b = 5, c = 6;

uint8_t R1 = a * b / c;

uint8_t R2 = a * b;
        R2 /= c;

cout << "R1: " << (int)R1 << '\n';
cout << "R2: " << (int)R2 << '\n';
```

```
Output:
    R1: 105
    R2: 20
```

boost::accumulators

```
accumulator_set<int, stats<tag::sum, tag::mean>> acc;

acc(1000);
acc(2000);
acc(2000);
acc(1000);

std::cout << "Sum:  " << sum(acc) << std::endl;
std::cout << "Mean: " << mean(acc) << std::endl;
```

```
Output:
    Sum:  6000
    Mean: 1500
```

User’s guide

sum *and variants*

For summing the samples, weights or variates. The default implementation uses the standard sum operation, but variants using the Kahan summation algorithm are also provided.

Result Type

`sample-type` for summing samples
`weight-type` for summing weights
`variate-type` for summing variates

Depends On

none

Variants

```
tag::sum
tag::sum_of_weights
tag::sum_of_variates<variate-type, variate-tag>
tag::sum_kahan (a.k.a. tag::sum(kahan))
tag::sum_of_weights_kahan (a.k.a. tag::sum_of_weights(kahan))
tag::sum_of_variates_kahan<variate-type, variate-tag>
```

Initialization Parameters

none

Accumulator Parameters

`weight` for summing weights
`variate-tag` for summing variates

Extractor Parameters

none

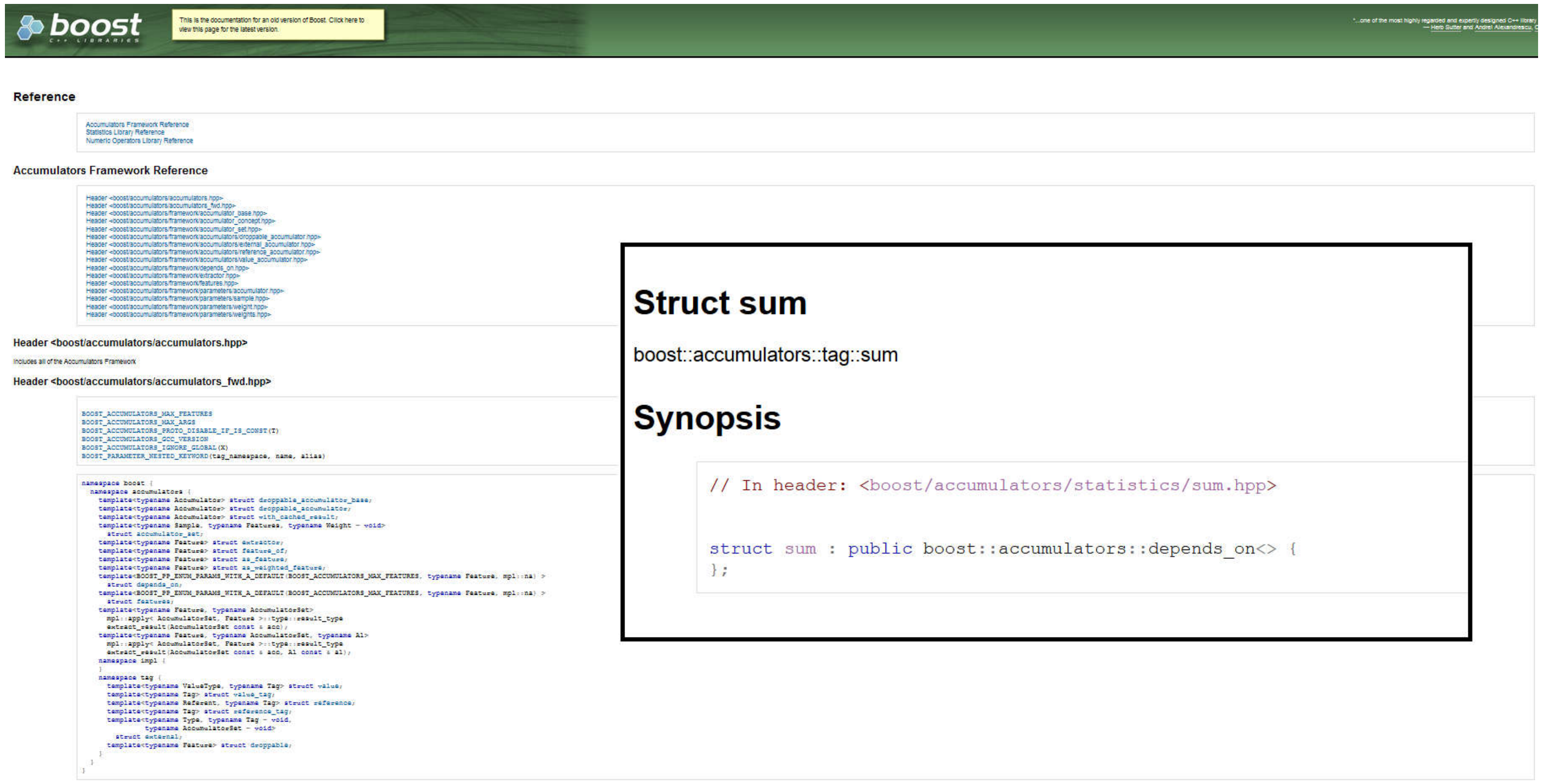
Accumulator Complexity

O(1). Note that the Kahan sum performs four floating-point sum operations per accumulated value, whereas the naive sum performs only one.

Extractor Complexity

O(1)

Reference



boost C++ LIBRARIES

This is the documentation for an old version of Boost. Click here to view this page for the latest version.

"...one of the most highly regarded and expertly designed C++ library
— Herb Sutter and Andrei Alexandrescu, C

Reference

Accumulators Framework Reference
Statistics Library Reference
Numeric Operators Library Reference

Accumulators Framework Reference

Header <boost/accumulators/accumulators.hpp>
Header <boost/accumulators/accumulators_fwd.hpp>
Header <boost/accumulators/framework/accumulator_base.hpp>
Header <boost/accumulators/framework/accumulator_concept.hpp>
Header <boost/accumulators/framework/accumulator_set.hpp>
Header <boost/accumulators/framework/accumulators/droppable_accumulator.hpp>
Header <boost/accumulators/framework/accumulators/external_accumulator.hpp>
Header <boost/accumulators/framework/accumulators/reference_accumulator.hpp>
Header <boost/accumulators/framework/accumulators/value_accumulator.hpp>
Header <boost/accumulators/framework/depends_on.hpp>
Header <boost/accumulators/framework/extractor.hpp>
Header <boost/accumulators/framework/features.hpp>
Header <boost/accumulators/framework/parameters/accumulator.hpp>
Header <boost/accumulators/framework/parameters/sample.hpp>
Header <boost/accumulators/framework/parameters/weight.hpp>
Header <boost/accumulators/framework/parameters/weights.hpp>

Header <boost/accumulators/accumulators.hpp>

Includes all of the Accumulators Framework

Header <boost/accumulators/accumulators_fwd.hpp>

```
BOOST_ACCUMULATORS_MAX_FEATURES
BOOST_ACCUMULATORS_MAX_ARGS
BOOST_ACCUMULATORS_PROTO_DISABLE_IF_IS_CONST(T)
BOOST_ACCUMULATORS_GCC_VERSION
BOOST_ACCUMULATORS_IGNORE_GLOBAL(X)
BOOST_PARAMETER_NESTED_KEYWORD(tag_namespace, name, alias)
```

```
namespace boost {
  namespace accumulators {
    template<typename Accumulator> struct droppable_accumulator_base;
    template<typename Accumulator> struct droppable_accumulator;
    template<typename Accumulator> struct with_cached_result;
    template<typename Sample, typename Features, typename Weight = void>
      struct accumulator_set;
    template<typename Feature> struct extractor;
    template<typename Feature> struct feature_of;
    template<typename Feature> struct as_feature;
    template<typename Feature> struct as_weighted_feature;
    template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
      struct depends_on;
    template<BOOST_PP_ENUM_PARAMS_WITH_A_DEFAULT(BOOST_ACCUMULATORS_MAX_FEATURES, typename Feature, mpl::na) >
      struct features;
    template<typename Feature, typename AccumulatorSet>
      mpl::apply< AccumulatorSet, Feature >::type::result_type
      extract_result(AccumulatorSet const & acc);
    template<typename Feature, typename AccumulatorSet, typename A1>
      mpl::apply< AccumulatorSet, Feature >::type::result_type
      extract_result(AccumulatorSet const & acc, A1 const & a1);
    namespace impl {
    }
    namespace tag {
      template<typename ValueType, typename Tag> struct value;
      template<typename Tag> struct value_tag;
      template<typename Referent, typename Tag> struct reference;
      template<typename Tag> struct reference_tag;
      template<typename Type, typename Tag = void,
               typename AccumulatorSet = void>
        struct external;
      template<typename Feature> struct droppable;
    }
  }
}
```

Struct sum

boost::accumulators::tag::sum

Synopsis

```
// In header: <boost/accumulators/statistics/sum.hpp>


struct sum : public boost::accumulators::depends_on<> {
};
```

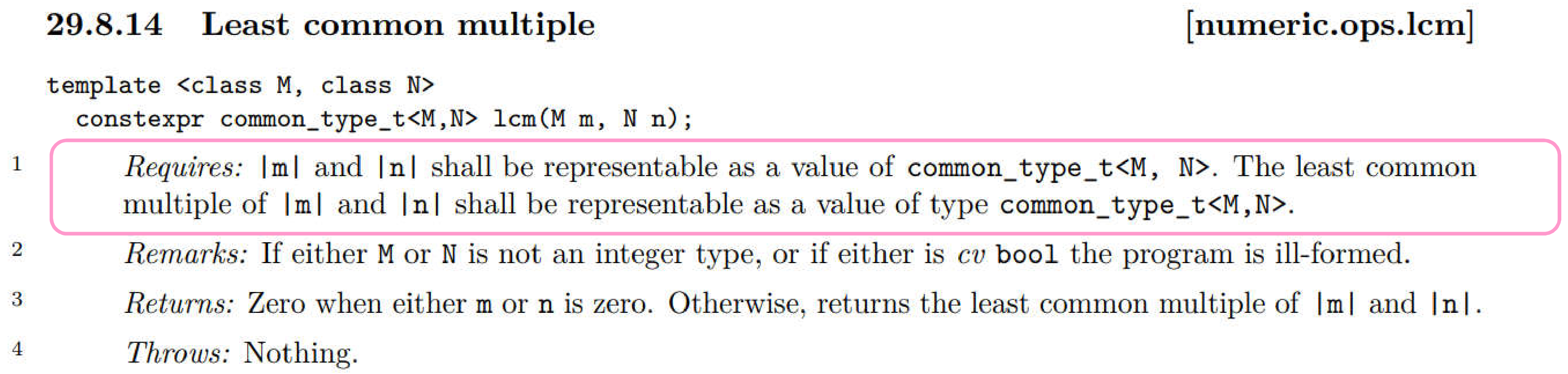
```
accumulator_set<int, stats<tag::sum, tag::mean>> acc;
//accumulator_set<int, stats<tag::sum, tag::mean(immediate)>> acc;

acc(1000'000'000);
acc(2000'000'000);
acc(2000'000'000);
acc(1000'000'000);

std::cout << "Sum:   " << sum(acc) << std::endl;
std::cout << "Mean:  " << mean(acc) << std::endl;
```

```
Output:
    Sum:  1705'032'704
    Mean: 4.26258E+8
 --------
 INT_MAX: 2147'483'647
Real sum: 6000'000'000
Real Mean: 1.5E+9
```

What about overflow?

```
template<typename Sample, typename Tag>
struct sum_impl : accumulator_base
{
    // ...
    template<typename Args>
    void operator ()(Args const &args)
    {
        // what about overflow?
        this->sum += args[parameter::keyword<Tag>::get()];
    }
    // ...
    Sample sum;
};
```

C++17 added GCD & LCM support



29.8.14 Least common multiple [numeric.ops.lcm]

```
template <class M, class N>
  constexpr common_type_t<M,N> lcm(M m, N n);
```

1 *Requires:* `|m|` and `|n|` shall be representable as a value of `common_type_t<M, N>`. The least common multiple of `|m|` and `|n|` shall be representable as a value of type `common_type_t<M,N>`.

2 *Remarks:* If either `M` or `N` is not an integer type, or if either is *cv* `bool` the program is ill-formed.

3 *Returns:* Zero when either `m` or `n` is zero. Otherwise, returns the least common multiple of `|m|` and `|n|`.

4 *Throws:* Nothing.

LCM

- lcm(65537, 65539) = **262‘147**
- Actually it’s 4'295'229'443
- or 0x1'0004'0003
- or 33 bits

29.8.2 Accumulate [accumulate]

```
template <class InputIterator, class T>
  T accumulate(InputIterator first, InputIterator last, T init);
template <class InputIterator, class T, class BinaryOperation>
  T accumulate(InputIterator first, InputIterator last, T init,
               BinaryOperation binary_op);
```

1 *Requires:* `T` shall meet the requirements of `CopyConstructible` (Table 24) and `CopyAssignable` (Table 26) types. In the range `[first, last]`, `binary_op` shall neither modify elements nor invalidate iterators or subranges.[281]

2 *Effects:* Computes its result by initializing the accumulator `acc` with the initial value `init` and then modifies it with `acc = acc + *i` or `acc = binary_op(acc, *i)` for every iterator `i` in the range `[first, last)` in order.[282]

Unsafe operations

- **<numeric> header**
  - reduce
  - inner_product
  - inclusive_scan
  - exclusive_scan
  - transform_reduce
  - partial_sum
  - transform_exclusive_scan
  - transform_inclusive_scan
  - adjacent_difference
- **<valarray> header**
  - T sum();
  - operator *=
  - operator /=
  - operator +=
  - operator -=

So how do you detect overflows?

- **The processor does it for you for free!**
- The standard provides

```
 imaxdiv_t imaxdiv(intmax_t number, intmax_t denom);
```

- But no add, subtract, multiply nor other division functions
- We can use compiler extensions, write our own assembly routines or simulate the operations in code

Addition and subtraction

```
bool add_of(int32_t sx, int32_t sy) {
    uint32_t x = sx;
    uint32_t y = sy;
    return bool(((~(x ^ y)) & ((x + y) ^ x)) >> 31);
}

bool add_of(uint32_t x, uint32_t y) {
    return bool(((x & y) | ((x | y) & ~(x + y))) >> 31);
}

bool sub_of(int32_t sx, int32_t sy) {
    uint32_t x = sx;
    uint32_t y = sy;
    return bool(((x ^ y) & ((x - y) ^ x)) >> 31);
}

bool sub_of(uint32_t x, uint32_t y) {
    return bool(((~x & y) | ((~x | y) & (x - y))) >> 31);
}
```

Multiplication

```
bool mul_of(int32_t sx, int32_t sy) {
    uint32_t x = sx;
    uint32_t y = sy;
    uint32_t m, n, t, z;

    m = nlz(x) + nlz(~x);
    n = nlz(y) + nlz(~y);

    if (m + n <= 30) return true;

    t = x * (y >> 1);
    if ((int32_t)t < 0) return true;

    z = t * 2;
    if (y & 1) {
        z = z + x;
        if (z < x) return true;
    }

    return false;
}
```

```
bool mul_of(uint32_t x, uint32_t y) {
    uint32_t m, n, t, z;

    m = nlz(x);
    n = nlz(y);

    if (m + n <= 31) return true;

    t = x * (y >> 1);
    if ((int32_t)t < 0) return true;

    z = t * 2;
    if (y & 1) {
        z = z + x;
        if (z < x) return true;
    }

    return false;
}
```

Multiplication

```
void mul(uint32_t x, uint32_t y, uint32_t p[2]) {
    uint32_t A = x >> 16;
    uint32_t B = x & 0xFFFF;
    uint32_t C = y >> 16;
    uint32_t D = y & 0xFFFF;

    uint32_t bd = B*D;
    uint32_t tmp = B*C;

    p[1] = A*C + (tmp >> 16);
    tmp = (tmp & 0xFFFF) + A*D;
    p[1] += tmp >> 16;
    tmp = (tmp & 0xFFFF) + (bd >> 16);
    p[1] += tmp >> 16;
    p[0] = (tmp << 16) | (bd & 0xFFFF);
}
```

```
void mul(int32_t x, int32_t y, dblint_t& p) {
    uint32_t s = (x >= 0) ? 0 : -y;
    uint32_t t = (y >= 0) ? 0 : -x;

    uint32_t up[2];

    mul(uint32_t(x), uint32_t (y), up);

    p.low = up[0];
    p.high = up[1] + s + t;
}
```

Division

```c
bool div_of(int32_t sx, int32_t sy) {
    return (sy == 0) || ((uint32_t)sx == 0x80000000u && sy == -1);
}

bool div_of(uint32_t x, uint32_t y) {
    (void)x;
    return y == 0;
}
```

```
void div(const uint16_t a[2], uint16_t b, uint16_t q[2], uint16_t& r)
{
    if (!a[1] && a[0] < b) {
        // the dividend is smaller the divisor
        r = a[0];
        q[0] = 0;
        q[1] = 0;
        return;
    }

    if (b & 0xFF00u) {
        uint8_t aa[5];
        int s = 0;    // shift needed for normalization

        while (!(b & 0x8000u)) {
            ++s;
            b <<= 1;
        }

        const uint8_t * tmp = (const uint8_t *)a;
        aa[0] = tmp[0] << s;
        aa[4] = tmp[3] >> (8-s);
        for (int i=1; i<4; ++i) {
            aa[i] = (tmp[i] << s) | (tmp[i-1] >> (8-s));
        }

        div_normalized(aa, b, q, r);

        r >>= s;
    }
    else {
        // the divisor fits in one half-word, special case
        div_by_half((const uint8_t*)a, uint8_t(b), q, r);
    }
}
```

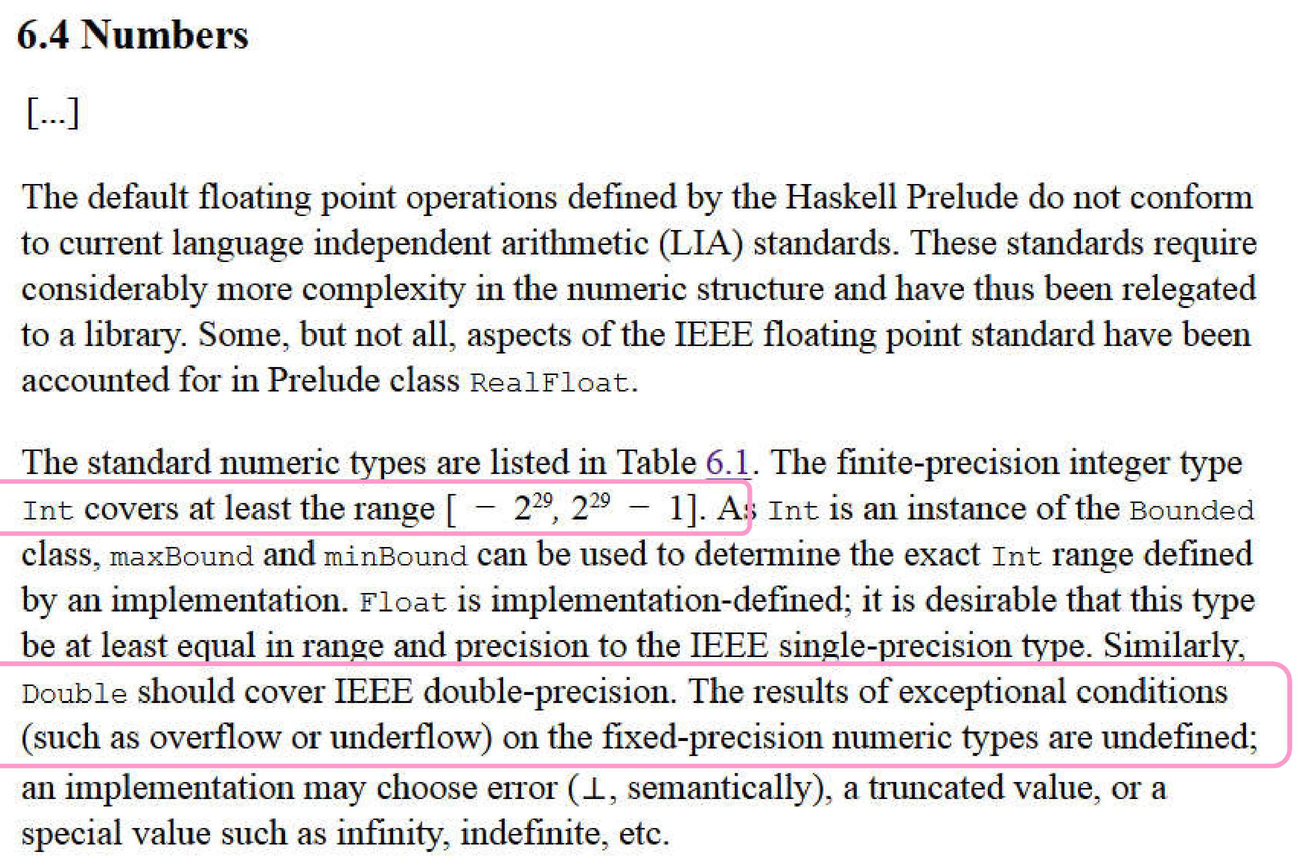
```
void div_by_half(const uint8_t aa[4], uint8_t bb0, uint16_t q[2], uint16_t& r) {
    r = 0;
    uint8_t* qq = (uint8_t*)q;

    for (int i = 3; i >= 0; i--) {
        qq[i] = ((r << 8) + aa[i]) / bb0;
        r = ((r << 8) + aa[i]) - qq[i] * bb0;
    }
}

void div_normalized(const uint8_t aa[5], uint16_t b, uint16_t q[2], uint16_t& r) {
    uint8_t qr[4];

    div_3_limbs_by_2(aa[4], aa[3], aa[2], b, qr);

    q[1] = (uint16_t(qr[1]) << 8) + qr[0];

    div_3_limbs_by_2(qr[3], qr[2], aa[1], b, qr);

    q[1] += qr[1];
    q[0] = uint16_t(qr[0]) << 8;

    div_3_limbs_by_2(qr[3], qr[2], aa[0], b, qr);

    uint16_t tmp = q[0];
    q[0] += (uint16_t(qr[1]) << 8) + qr[0];

    if (tmp > q[0]) q[1]++; // handle the overflow

    r = (uint16_t(qr[3]) << 8) + qr[2];
}
```

```
void div_3_limbs_by_2(uint8_t a1, uint8_t a2, uint8_t a3, uint16_t b, uint8_t qr[4])
{
    const uint8_t* bb = (const uint8_t*)&b;
    uint16_t tmp = (uint16_t(a1) << 8) + a2;
    uint16_t qe = tmp / bb[1];
    uint16_t c = tmp - qe * bb[1];
    uint16_t d = qe * bb[0];
    uint16_t r = (c << 8) + a3;

    if (r < d) { // qe is too large by at least one, max 2
        qe--;
        r += b; // may overflow, the test bellow is checking for this

        if (r >= b && r < d) {
            qe--;
            r += b; // may overflow by 1 bit, but the line 'r -= d' will still yield
        }           // the correct result since it will 'borrow' one bit
    }

    r -= d;
    qr[0] = uint8_t(qe);
    qr[1] = qe >> 8;
    qr[2] = r & 0xFF;
    qr[3] = r >> 8;
}
```

Let’s fix accumulate

```
T accumulate(It first, It last, T val) {
    int overflow = 0;

    for (; first != last; ++first)
        overflow |= add_of(val, *first);

    if (overflow) {
        // ... error
    }

    return (val);
}
```

Haskell



6.4 Numbers

[...]

The default floating point operations defined by the Haskell Prelude do not conform to current language independent arithmetic (LIA) standards. These standards require considerably more complexity in the numeric structure and have thus been relegated to a library. Some, but not all, aspects of the IEEE floating point standard have been accounted for in Prelude class `RealFloat`.

The standard numeric types are listed in Table 6.1. The finite-precision integer type `Int` covers at least the range $[-2^{29}, 2^{29}-1]$. As `Int` is an instance of the `Bounded` class, `maxBound` and `minBound` can be used to determine the exact `Int` range defined by an implementation. `Float` is implementation-defined; it is desirable that this type be at least equal in range and precision to the IEEE single-precision type. Similarly, `Double` should cover IEEE double-precision. The results of exceptional conditions (such as overflow or underflow) on the fixed-precision numeric types are undefined; an implementation may choose error ($\perp$, semantically), a truncated value, or a special value such as infinity, indefinite, etc.

Concluding remarks

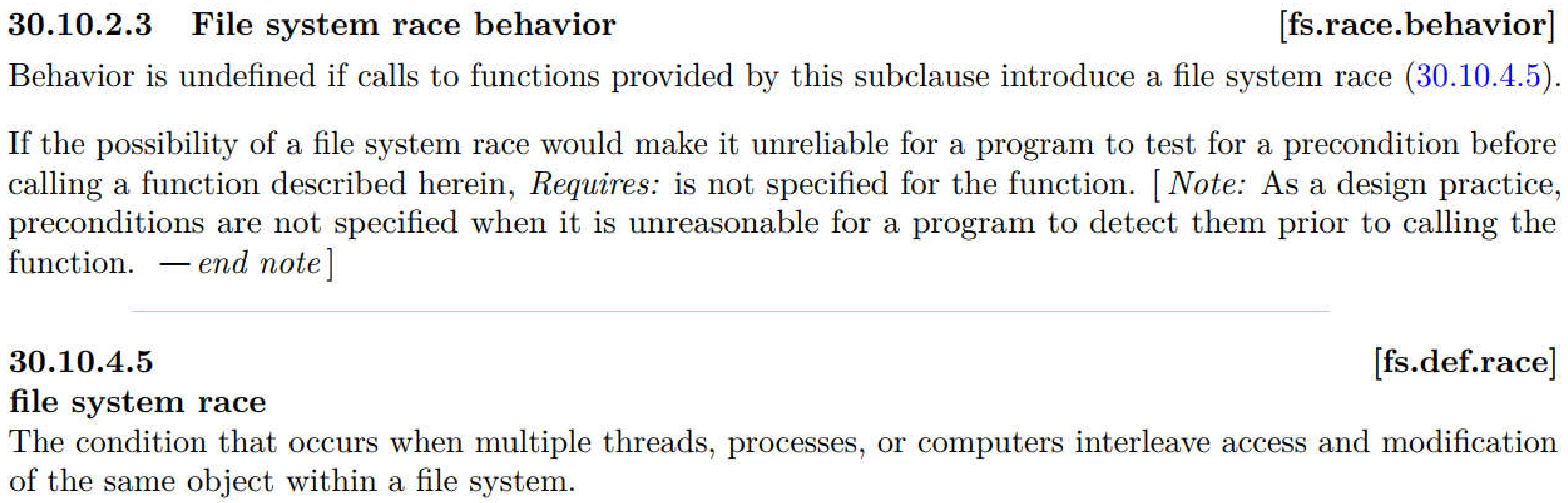
- The standard library could help us by providing add, sub, mul and div variants
- Abstracting away essential details leads to incorrect code and APIs. (LSP for templates)
- It is very easy to create unusable interfaces
- The documentation is part of the API. If any pre-condition or behavior changes, the API itself has changed
- We need libraries that provide reliable, safe and portable implementations and APIs

Questions

- Hacker's Delight 2nd Ed. by Henry S. Warren, Jr., ISBN 0-321-84268-5
- The Art of Computer Programming: Seminumerical Algorithms by Donald Knuth
- Burnikel C., Ziegler J., "Fast Recursive Division", MPI-I-98-1-022
- Hansen, Per Brinch, "Multiple-Length Division Revisited: A Tour of the Minefield"
- https://www.haskell.org/onlinereport/haskell2010/haskellch6.html#x13-1350006.4

azlatepodani@gmail.com

Is this a realistic precondition?



30.10.2.3 File system race behavior **[fs.race.behavior]**

Behavior is undefined if calls to functions provided by this subclause introduce a file system race (30.10.4.5).

If the possibility of a file system race would make it unreliable for a program to test for a precondition before calling a function described herein, *Requires:* is not specified for the function. [*Note:* As a design practice, preconditions are not specified when it is unreasonable for a program to detect them prior to calling the function. — *end note*]

30.10.4.5 **[fs.def.race]**
file system race
The condition that occurs when multiple threads, processes, or computers interleave access and modification of the same object within a file system.

Bugs in the wild

- CVE-2016-5223 Integer overflow in […] Google **Chrome** prior to 55.0.2883.75…
- CVE-2017-14051 An integer overflow in […] the **Linux kernel** through 4.12.10…
- CVE-2017-7529 **Nginx** versions […] are vulnerable to integer overflow…
- CVE-2017-3738 There is an overflow bug in the AVX2 Montgomery multiplication procedure […] **OpenSSL**…