Doc No: N1541=03-0124
Date: 13 Nov 2003
Reply to: Matt Austern
austern@apple.com

Library Extension Technical Report — Issues List

Revision 1: Post-Kona

1 TR Introduction issues

1.1 How to disable TR features

Section: 1 [tr.intro]
Submitter: Matt Austern
Status: NAD

The TR says that implementers should not enable the TR by default, and should hide TR features more thoroughly than just putting them in another namespace. It's vague on exactly what implementers should do: have files in another directory (perhaps even shadow headers, like an alternate version of <functional>), or use a macro, or something else. Should we be more specific?

Resolution:
The LWG decided that the current text is satisfactory.

1.2 Feature test macros for the TR

Section: 1 [tr.intro]
Submitter: Beman Dawes
Status: New

How can users determine whether or not a particular compiler/library implementation supports the components described in the library extension TR? Should we have a coarse-grained macro (yes or not), or should we have a fine-grained facility so users can perform feature tests for individual pieces?

2 Smart pointer issues

2.1 shared_ptr constructor from auto_ptr missing postcondition

Submitter: Beman Dawes
Section: 2.2.3 [tr.util.smartptr.shared.const]
Status: Voted into the TR

For

```
template <class Y> shared_ptr<Y>(auto_ptr<Y> & r)
```

The Postcondition clause says:

```
use_count() == 1
```

Resolution:
change it to

```
use_count() == 1 && r.get() == 0
```

2.2 Error in shared_ptr constructor

Submitter: Pete Becker
Section: 2.2.3 [tr.util.smartptr.shared.const]
Paper: c++std-lib-11461, 11463-11510, 11512-11524
Status: Closed

template<class Y> explicit shared_ptr(Y *p);
template <class Y, class D> shared_ptr(Y *, D d);
template <class Y> shared_ptr<auto_ptr<Y> & r);

The Effects clauses for the first two ctors say:

Constructs a shared_ptr that owns the pointer p [and the deleter d].

and their Postconditions clauses say:

use_count() == 1 && get() == p

Similarly, the Effects clause for the third ctor says:

Constructs a shared_ptr that stores and owns r.release()

and the Postcondition clause says:

use_count == 1

Issues:

- Is this the correct behavior when p or r.release() is a null pointer? Consistency with the default constructor would suggest that use_count() == 0 for a null pointer, i.e. the result is an empty shared_ptr.
- If use_count() should be 0, this raises the lesser issue of whether smart_ptr(null, Dtor) should remember _Dtor, or should be equivalent to smart_ptr(). I'm pretty sure I prefer the latter, 'cause it's the way I've implemented it. (It's also simpler and more efficient to treat all null pointers the same way).

Resolution:
Discussed at Kona. There are several ways of phrasing this issue: Do we reference-count null pointers? Are null pointers a special case? What is the deleter argument good for? There wasn't consensus for changing what the TR already says, but it was agreed that this exposed another issue (2.3, see below).

2.3 shared_ptr equality and operator<

Submitter: Beman Dawes
Section: [tr.util.smartptr.shared]
Status: New

When two shared_ptrs p1 and p2 are constructed from the same underlying pointer, the behavior of operator== and operator< is surprising. We will have p1 == p2, but also either p1 < p2 or p1 > p2. We thus violate the usual trichotomy condition. For example, if you have a whole bunch of shared_ptrs in a set, you can't search for it by constructing a new shared_ptr.

It may seem that this is irrelevant because it's never correct to have two shared_ptrs with the same underlying pointer, but that's wrong. It's valid in two cases: (1) when the underlying pointer is null; or (2) when you're using a user-defined deleter object that doesn't do deletion.

3 Type traits issues

3.1 Use of Language in type transformations

Submitter: Pete Becker
Status: Voted into the TR

See N1519 for discussion of the issue.

Resolution:
Accept the proposed resolution for N1519. *[but editorial change: also add a non-normative note pointing out what it means for cv-qualified types]*

3.2 Why three headers?

Submitter: Pete Becker
Status: Voted into the TR

Three headers seems excessive. Why not put them all into <type_traits>? That would simplify things for users, who wouldn't have to remember which of the three headers defines the template they're interested in. Currently, <type_traits> has 33 templates (not counting helpers), <type_compare> has 3, and <type_transform> has 11. The classification is reasonable in itself, but I don't think it's particularly helpful.

A number of people expressed support for one header on the LWG reflector.

Resolution: Combine the three type traits headers into a single header named <type_traits>.

3.3 Is integral_constant an implementation detail?

Submitter: Pete Becker
Status: NAD

See N1519 for discussion of the issue.

Resolution:
NAD. We accepted several changes that require integral_constant to be exposed explicitly.

3.4 Revising the Unary Type Traits Requirements

Submitter: John Maddock
Status: voted into the TR

See N1519 for discussion of the issue.

Resolution: Accept the proposed resolution from N1519.

3.5 New type trait: alignment_of

Submitter: John Maddock
Status: voted into the TR

See N1519 for discussion of the issue.

Resolution: Accept the proposed resolution from N1519.

3.6 New type trait: has_virtual_destructor

Submitter: John Maddock
Status: voted into the TR

See N1519 for discussion of the issue.

Resolution: Accept the proposed resolution from N1519, but add a proviso that **false** is the fallback position if the compiler can't determine an exact answer.

3.7 New type trait: is_safely_destructible

Submitter: Bronek Kozicki
Status: NAD

See N1508 for discussion of the issue.

Resolution: The LWG decided not to accept this proposal. If we accepted it, it would be better for the template to have two parameters: can class **D** be safely destroyed via a pointer to class **B**? But as is, the trait seems too high level: it answers a complicated compound question, not an atomic question.

3.8 New type trait: rank

Submitter: John Maddock
Status: Open

See N1519 for discussion of the issue.

Resolution:

Discussed at Kona. The LWG wasn't sure whether this was useful; the few people who could use it reliably for metaprogramming would probably find it just as easy to write it themselves.

3.9 New type trait: dimension

Submitter: John Maddock
Status: Open

See N1519 for discussion of the issue.

Resolution:
Discussed at Kona. Same status as **rank**: the LWG wasn't sure whether this was useful.

3.10 New type trait: aligned_storage

Submitter: John Maddock
Status: Voted into the TR

See N1519 for discussion of the issue.

Resolution:
Accept the proposed resolution from N1519, but say "unspecified" instead of "implementation defined."

3.11 New type trait: remove_all_dimensions

Submitter: John Maddock
Status: Voted into the TR

See N1519 for discussion of the issue.

Resolution:
Accept the proposed resolution from N1519.

3.12 Conversion of traits to integral_constant

Submitter: Dave Abrahams
Status: New

Every traits class **X** has a nested typedef **type**, and has a conversion operator, **operator type() const**. Automatic conversions are useful and important, but a conversion operator is the wrong way to do it. Instead, we should say that **X** inherits from **type**. This would be consistent with actual implementation practice.

3.13 is_base_and_derived<X,X>

Submitter: Dave Abrahams
Status: New

Currently, `is_base_and_derived<X,Y>` returns false when X and Y are the same. This is technically correct (X isn't its own base class), but it isn't useful. The definition should be loosened to return true when X and Y are the same, even when the type isn't actually a class.

4 Random number generator issues

4.1 Confusing Text in Description of v.min()

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

In "Uniform Random Number Requirements" the text says that v.min() "Returns ... l where l is ...". This is the letter ell, which is too easily confused with the numeral one. Can we change it to something less confusing, like "lim"?

Resolution:
Change the first sentence of the description of v.min() in 5.1.1 [tr.rand.req], Table 5.2 (Uniform random number generator requirements) from:

Returns some l where l is less than or equal to all values potentially returned by operator().

to:

Returns a value that is less than or equal to all values potentially returned by operator().

4.2 Confusing and Incorrect Text in Description of v.max()

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

In "Uniform Random Number Requirements" the text says that v.max() "returns l where l is less than or equal to all values...". Should this be "greater than or equal to"? And similarly, should "strictly less than" be "strictly greater than."?

Resolution:
Change the first sentence of the description of `v.max()` in 5.1.1 [tr.rand.req], Table 5.2 (Uniform random number generator requirements) from:

If `std::numeric_limits<T>::is_integer`, returns `l` where `l` is less than or equal to all values potentially returned by `operator()`, otherwise, returns `l` where `l` is strictly less than all values potentially returned by `operator()`.

to:

If `std::numeric_limits<T>::is_integer`, returns a value that is greater than or equal to all values potentially returned by `operator()`, otherwise, returns a value that is strictly greater than all values potentially returned by `operator()`.

4.3 Table "Number Generator Requirements" Unnecessary

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The table "Number Generator Requirements" has only one entry: `X::result_type`. While it's true that random nunber generators and random distributions have this member, it doesn't seem like a useful basis for classification -- there's nothing in the proposal that depends on knowing that some type satisfies this requirement. I think the specification of `X::result_type` should be in "Uniform Random Number Generator Requirements" and in "Random Distribution Requirements."

Resolution:
Copy the description of `X::result_type` from 5.1.1 [tr.rand.req], Table 5.1 (Number generator requirements) to 5.1.1 [tr.rand.req], Table 5.2 (Uniform random number generator requirements) and to 5.1.1 [tr.rand.req], Table 5.4 (Random distribution requirements) and remove 5.1.1 [tr.rand.req], Table 5.1 (Number generator requirements).

4.4 Should a variate_generator Holding a Reference Be Assignable?

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The third paragraph says, in part:

> Specializations of `variate_generator` satisfy the requirements of CopyConstructible. They also satisfy the requirements of Assignable unless the template parameter `Engine` is of the form`U&`.

This looks like an implementation artifact. Is there a reason that variate_generators whose engine type is a reference should not be copied?

Resolution:
Change the first two sentences of the third paragraph of 5.1.3 [tr.rand.var] from:

> Specializations of `variate_generator` satisfy the requirements of CopyConstructible. They also satisfy the requirements of Assignable unless the template parameter `Engine` is of the form `U&`.

to:

> Specializations of `variate_generator` satisfy the requirements of CopyConstructible and Assignable. [*Note:* If the template parameter `Engine` is of reference type it is the reference, not the object referred to, that is copied. *—End Note]*

4.5 Normal Distribution Incorrectly Specified

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

For `normal_distribution`, the paper says that the probability density function is`1/sqrt(2*pi*sigma) * exp(- (x - mean)^2 / (2 * sigma^2))`. The references I've seen have a different initial factor, using `1/(sqrt(2*pi) * sigma)`. That is, `sigma` is outside the square root.

Resolution:
Change the first paragraph of 5.1.7.8 [tr.rand.dist.norm] from:

> A `normal_distribution` random distribution produces random numbers `x` distributed

with probability density function $(1/sqrt(2*pi*sigma))e^{-(x-mean)2/(2*sigma2)}$, where `mean` and `sigma` are the parameters of the distribution.

to:

A `normal_distribution` random distribution produces random numbers `x` distributed with probability density function $(1/(sqrt(2*pi)*sigma))e^{-(x-mean)2/(2*sigma2)}$, where `mean` and `sigma` are the parameters of the distribution.

4.6 Should Random Number Initializers Take Iterators by Reference or by Value?

Submitter: Pete Becker (see N1535)
Status: Open

See N1535 for a full discussion. Summary: when engines are seeded, the seed may be arbitrarily large. For compound engines we use a range where the first iterator is taken by reference and updated. This is an unconventional interface and will invite bugs. The obvious solution would be to have a function that takes iterators `first` and `last` by value and returns the updated version of first. However, this is an awkward solution for constructors. One possibility would be to abandon range constructors, and rely instead on two-phase initialization where the iterators are passed to a member function.

Resolution: Discussed at Kona, no decision. The status quo is awkward, but we don't have a better solution yet. Pete and Jens will work on this and will propose a solution for Sydney.

4.7 Are Global Operators Overspecified?

Submitter: Pete Becker (see N1535)
Status: Open

See N1535 for a full discussion. Summary: Do we literally want to require the existence of a namespace-scope `operator==`, or do we just want to say that when `x` and `y` are engines, `x == y` is required to work?

Resolution: Discussed at Kona, general agreement that we don't want to require a specific signature. Pete and Jens will provide wording along these lines.

4.8 Should the Template Arguments Be Restricted to Built-in Types?

Submitter: Pete Becker (see N1535)
Status: Voted into the TR.

See N1535 for a full discussion. Summary: Generators and distributions are parameterized on arithmetic types. The TR tries to allow user defined number-like types, but it's very hard to get that sort of thing right. We should restrict it to the built-in arithmetic types.

Resolution:

Replace in 5.1.1 [tr.rand.req], last paragraph

Furthermore, a template parameter named RealType shall denote a type that holds an approximation to a real number. This type shall meet the requirements for a numeric type

(26.1 [lib.numeric.requirements]), the binary operators +, -, *, / shall be applicable to it, a conversion from double shall exist, and function signatures corresponding to those for type double in subclause 26.5 [lib.c.math] shall be available by argument-dependent lookup (3.4.2 [basic.lookup.koenig]). [Note: The built-in floating-point types float and double meet these requirements.]

by

Furthermore, the effect of instantiating a template that has a template type parameter named`RealType` is undefined unless that type is one of `float`, `double`, or `long double`.

Delete from 5.1.7 [tr.rand.dist]

A template parameter named IntType shall denote a type that represents an integer number. This type shall meet the requirements for a numeric type (26.1 [lib.numeric.requirements]), the binary operators +, -, *, /, % shall be applicable to it, and a conversion from int shall exist. [Footnote: The built-in types int and long meet these requirements.]

...

No function described in this section throws an exception, unless an operation on values of IntType or RealType throws an exception. [Note: Then, the effects are undefined, see [lib.numeric.requirements].]

Add after 5.1.1 [tr.rand.req], last paragraph

The effect of instantiating a template that has a template type parameter named `IntType` is undefined unless that type is one of `short`, `int`, `long`, or their unsigned variants.

The effect of instantiating a template that has a template type parameter named `UIntType` is undefined unless that type is one of `unsigned short`, `unsigned int`, or `unsigned long`.

4.9 Do Engines Need Type Arguments?

Submitter: Pete Becker (see N1535)
Status: Open

See N1535 for a discussion. Summary: engines are parameterized by type, but this is pretty much redundant. The appropriate type can be deduced from the template arguments.

Resolution: Discussed at Kona. No consensus that this change would be a good idea.

4.10 Unclear Complexity Requirements for variate_generator

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The specification for `variate_generator` says

Specializations of `variate_generator` satisfy the requirements of CopyConstructible. They also satisfy the requirements of Assignable unless the template parameter `Engine` is of

the form`U&`. The complexity of all functions specified in this section is constant. No function described in this section except the constructor throws an exception.

Taken literally, this isn't implementable. `operator()` calls the underlying distribution's `operator()`, whose complexity isn't directly specified. The distribution's `operator()` makes an amortized constant number of calls to the generator's `operator()`, whose complexity is, again, amortized constant. So the complexity of`variate_generator::operator()` ought to also be amortized constant.

`variate_generator` also has a constructor that takes an engine and a distribution by value, and uses their respective copy constructors to create internal copies. There are no complexity constraints on those copy constructors, but given that the default constructor for an engine has complexity O(size of state), it seems likely that an engine's copy constructor would also have complexity O(size of state). This means that`variate_generator`'s complexity is at best O(size of engine's state), not constant.

I suspect that what was intended was that these functions would not introduce any additional complexity, that is, their complexity is the "larger" of the complexities of the functions that they call.

Resolution:
Replace in 5.1.3 [tr.rand.var]
The complexity of all functions specified in this section is constant.
by
Except where otherwise specified, the complexity of all functions specified in this section is constant.

Add for `variate_generator(engine_type e, distribution_type d)`
Complexity: Sum of the complexities of the copy construtors of `engine_type` and`distribution_type`.

Add for `result_type operator()()`
Complexity: Amortized constant.

Add for `result_type operator()(T value)`
Complexity: Amortized constant.

4.11 xor_combine Over-generalized?

Submitter: Pete Becker (see N1535)
Status: Editorial

For an `xor_combine` engine, is there ever a case where both `s1` and `s2` would be non-zero? Seems like this would produce non-random values, because the low bits (up to the smaller of the two shift values) would all be 0.

If at least one has to be 0, then we only need one shift value, and the definition might look more like this:

```
template <class _Engine1, class _Engine2, int _Shift = 0>
```

...

with the output being `(_Eng1() ^ (_Eng2() << _Shift))`.

Resolution: Discussed at Kona. The LWG felt that this interface is still the simplest. The right solution is to add a non-normative note advising users that only one of these parameters should be nonzero. The project editor is directed to add that note.

4.12 xor_combine::result_type Incorrectly Specified

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

`xor_combine` has a member
```
 typedef typename base_type::result_type result_type;
```

However, it has no type named `base_type`, only `base1_type` and `base2_type`. So, what should `result_type` be?

Resolution:
In 5.1.4.6 [tr.rand.eng.xor] replace
    typedef typename base_type::result_type result_type;
by
    typedef /* see below */ result_type;

and add at the end of the paragraph below the class definition
    The member `result_type` is defined to that type ofUniformRandomNumberGenerator1::result_type andUniformRandomNumberGenerator2::result_type that provides the most storage [basic.fundamental].

4.13 subtract_with_carry's IntType Overpecified

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The IntType for subtract_with_carry "shall denote a signed integral type large enough to store values up to `m - 1`." The implementation subtracts two values of that type, and if the result is `< 0` it adds back the `m`, which makes the result non-negative. In fact, this also works for unsigned types, with just a small change in the implementation: instead of testing whether the result is `< 0` you test whether it's `< 0` or greater than or equal to `m`. This works because unsigned arithmetic wraps, and it makes the template a bit easier to use.

I suggest that we loosen the constraint to allow signed and unsigned types. Thus the constraint would read "shall denote an integral type large enough to store values up to `m - 1`."

Resolution:
In 5.1.4.3 [tr.rand.eng.sub], replace
    The template parameter `IntType` shall denote a signed integral type large enough to store

values up to `m`-1.

by

The template parameter `IntType` shall denote an integral type large enough to store values up to `m`.

4.14 subtract_with_carry_01::seed(unsigned) Missing Constaint

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The specification for `subtract_with_carry::seed(IntVal)` has a *Requires* clause which requires that the argument be greater than 0. This member function needs the same constraint.

Resolution:
Add:

Requires: `value > 0`

to the description of `subtract_with_carry_01::seed(unsigned)` in 5.1.4.4 [tr.rand.eng.sub1]. (See resolution of issue 4.19, which also affects the wording in this area.)

4.15 subtract_with_carry_01::seed(unsigned) Produces Bad Values

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

`subtract_with_carry_01::seed(unsigned int)` uses a linear congruential generator to produce initial values for the fictitious previously generated values. These values are generated as `(y(i)*2^-w) mod 1`. The linear congruential generator produces values in the range `[0, 2147483564)`, which are at most 31 bits long. If the template argument `w` is greater than 31 the initial values generated by seed will all be rather small, and the first values produced by the generator will also be rather small. The Boost implementation avoids this problem by combining values from the linear congruential generator to produce longer values when `w` is larger than 32. Should we require something more like that?

Resolution:
In 5.1.4.4 [tr.rand.eng.sub1] replace

```
void seed(unsigned int value = 19780503)
```

Effects: With a linear congruential generator l(i) having parameters m = 2147483563, a = 40014, c = 0, and l(0) = `value`, sets x(-r) ... x(-1) to (l(1)*2-w) mod 1 ... (l(r)*2-w) mod 1, respectively. If x(-1) == 0, sets carry(-1) = 2-w, else sets carry(-1) = 0.

Complexity: O(r)

With

```
void seed(unsigned long value = 19780503ul)
```

Effects: With n=(w+31)/32 (rounded downward) and given an iterator range `[first, last)` that refers to the sequence of values lcg(1) ... lcg(n*r) obtained from a linear congruential generator lcg(i) having parameters mlcg = 2147483563, alcg = 40014, clcg = 0, and lcg(0) =`value`, invoke `seed(first,last)` .

Complexity: O(r*n)

4.16 subtract_with_carry_01::seed(unsigned) Argument Type Too Small

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

`subtract_with_carry_01::seed(unsigned)` has a default argument value of 19780503, which is too large to fit in a 16-bit unsigned int. Should this argument be unsigned long, to ensure that it's large enough for the default?

Resolution:
In 5.1.4.2 [tr.rand.eng.mers], change the signature of a constructor and a seed function from

```
explicit mersenne_twister(result_type value);
void seed(result_type value);
```

to

```
explicit mersenne_twister(unsigned long value);
void seed(unsigned long value);
```

In 5.1.4.3 [tr.rand.eng.sub], change the signature of a constructor and a seed function from

```
explicit subtract_with_carry(IntType value);
void seed(IntType value = 19780503);
```

to

```
explicit subtract_with_carry(unsigned long value);
void seed(unsigned long value = 19780503ul);
```

In 5.1.4.4 [tr.rand.eng.sub1], change the signature of a constructor and a seed function from

```
subtract_with_carry_01(unsigned int value);
void seed(unsigned int value = 19780503);
```

to:

```
subtract_with_carry_01(unsigned long value);
void seed(unsigned long value = 19780503ul);
```

4.17 subtract_with_carry::seed(In&, In) Required Sequence Length Too Long

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

For both `subtract_with_carry::seed(In& first, In last)` and`subtract_with_carry_01::seed(In& first, In last)` the proposal says: "With `n=w/32+1` (rounded downward) and given the values `z0 ... zn*r-1`." The idea is to use `n unsigned long` values to generate each of the initial values for the generator, so `n` should be the number of 32-bit words needed to provide `w`bits. Looks like it should be "`n=(w+31)/32`". As currently written, when `w` is 32, the function consumes two 32-bit values for each value that it generates. One is sufficient.

Resolution:
Change
With `n=w/32+1` (rounded downward) and given the values `z0 ... zn*r-1`

to

With `n=(w+31)/32` (rounded downward) and given the values `z0 ... zn*r-1`

in the description of `subtract_with_carry::seed(In& first, In last)` in 5.1.4.3 [tr.rand.eng.sub] and in the description of `subtract_with_carry_01::seed(In& first, In last)` in 5.1.4.4 [tr.rand.eng.sub1].

4.18 linear_congruential -- Giving Meaning to a Modulus of 0

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

Some linear congruential generators using an integral type `_Ty` also use a modulus that's equal to `numeric_limts<_Ty>::max() + 1` (e.g. 65536 for a 16-bit unsigned int). There's no way to write this value as a constant of the type `_Ty`, though. Writing it as a larger type doesn't work, because the linear_congruential template expects an argument of type `_Ty`, so you typically end up with a value that looks like 0.

On the other hand, the current text says that the effect of specifying a modulus of 0 for`linear_congruential` is implementation defined. I decided to use 0 to mean `max()+1`, as did the Boost implementation. (Internally, the implementation of `mersenne_twister` needs a generator with a modulus like this). Seems to me this is a reasonable choice, and one that users ought to be able to rely on. Is there some other meaning that might reasonably be ascribed to it, or should we say that a modulus of 0 means`numeric_limits<_Ty>::max() + 1` (suitably type-cast)?

Resolution:
Replace in 5.1.4.1 [tr.rand.eng.lcong], in the paragraph after the class definition

If the template parameter m is 0, the behaviour is implementation-defined.

by

If the template parameter m is 0, the modulus m used throughout this section is`std::numeric_limits<IntType>::max()` plus 1. [*Note*: The result is not representable as a value of type `IntType`. —*end note*]

4.19 linear_congruential::seed(IntType) -- Modify Seed Value When c == 0?

Submitter: Pete Becker (see N1535)
Status:

When c == 0 you get a generator with a slight quirk: if you seed it with 0 you get 0's forever; if you seed it with a non-0 value you never get 0. The first path, of course, should be avoided. The proposal does this by imposing a requirement on `seed(IntType x0)`, requiring that `c > 0 || (x0 % m) > 0`. The boost implementation uses asserts to check this condition. The only reservation I have about this is that it can only be checked at runtime, when the only suitable action is, probably, to abort. An alternative would be to force a non-0 seed in that case (perhaps 1, for no particularly good reason). I think the second alternative is marginally better, and I

suggest we change this requirement to impose a particular seed value when a user passes 0 to a generator with `c == 0`.

Resolution:
Replace in 5.1.4.1 [tr.rand.eng.lcong]

```
explicit linear_congruential(IntType x0 = 1)
```

Requires: c > 0 || (x0 % m) > 0
Effects: Constructs a linear_congruential engine with state x(0) := x0 mod m.

```
void seed(IntType x0 = 1)
```

Requires: c > 0 || (x0 % m) > 0
Effects: Sets the state x(i) of the engine to x0 mod m.

```
template linear_congruential(In& first, In last)
```

Requires: c > 0 || *first > 0
Effects: Sets the state x(i) of the engine to *first mod m.
Complexity: Exactly one dereference of *first.

by

```
explicit linear_congruential(IntType x0 = 1)
```

Effects: Constructs a `linear_congruential` engine and invokes `seed(x0)`.

```
void seed(IntType x0 = 1)
```

Effects: If `c` mod `m` = 0 and `x0` mod `m` = 0, sets the state x(i) of the engine to 1 mod `m`, else sets the state x(i) of the engine to `x0` mod `m`.

```
template linear_congruential(In& first, In last)
```

Effects: If `c` mod `m` = 0 and `*first` mod `m` = 0, sets the state x(i) of the engine to 1 mod `m`, else sets the state x(i) of the engine to `*first` mod `m`.
Complexity: Exactly one dereference of `*first`.

Replace in 5.1.4.2 [tr.rand.eng.mers]

```
void seed()
```

Effects: Invokes seed(4357).

```
void seed(result_type value)
```

Requires: value > 0
Effects: With a linear congruential generator l(i) having parameters m_l = 232, a_l = 69069, c_l = 0, and l(0) = value, sets x(-n) ... x(-1) to l(1) ... l(n), respectively.
Complexity: O(n)

by

```
void seed()
```

Effects: Invokes **seed(0)**.

```
void seed(result_type value)
```

Effects: If value == 0, sets value to 4357. In any case, with a linear congruential

generator lcg(i) having parameters m_{lcg} = 232, alcg = 69069, c_{lcg} = 0, and lcg(0) = `value`, sets x(-n) ... x(-1) to lcg(1) ... lcg(n), respectively.
Complexity: O(n)

Replace in 5.4.1.3 [tr.rand.eng.sub]

```
void seed(unsigned int value = 19780503)
```

Requires: `value > 0`
Effects: With a linear congruential generator l(i) having parameters ml = 2147483563, al = 40014, cl = 0, and l(0) = `value`, sets x(-r) ... x(-1) to l(1) mod m ... l(r) mod m, respectively. If x(-1) == 0, sets carry(-1) = 1, else sets carry(-1) = 0.
Complexity: O(r)

by

```
void seed(unsigned long value = 19780503ul)
```

Effects: If value == 0, sets value to 19780503. In any case, with a linear congruential generator lcg(i) having parameters m_{lcg} = 2147483563, a_{lcg} = 40014, c_{lcg} = 0, and lcg(0) =`value`, sets x(-r) ... x(-1) to lcg(1) mod m ... lcg(r) mod m, respectively. If x(-1) == 0, sets carry(-1) = 1, else sets carry(-1) = 0.
Complexity: O(r)

Replace in 5.4.1.4 [tr.rand.eng.sub1]

```
void seed(unsigned int value = 19780503)
```

Effects: With a linear congruential generator l(i) having parameters m = 2147483563, a = 40014, c = 0, and l(0) = `value`, sets x(-r) ... x(-1) to (l(1)*2-w) mod 1 ... (l(r)*2-w) mod 1, respectively. If x(-1) == 0, sets carry(-1) = 2-w, else sets carry(-1) = 0.
Complexity: O(r)

by

```
void seed(unsigned long value = 19780503ul)
```

Effects: If value == 0, sets value to 19780503. In any case, with a linear congruential generator lcg(i) having parameters m_{lcg} = 2147483563, a_{lcg} = 40014, c_{lcg} = 0, and lcg(0) =`value`, sets x(-r) ... x(-1) to lcg(1) mod m ... lcg(r) mod m, respectively. If x(-1) == 0, sets
Complexity: O(r)

4.20 linear_congruential -- Should the Template Arguments Be Unsigned?

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

This template takes three numeric arguments, `a`, `c`, and `m`, whose type is `IntType`. `IntType`is an integral type, possibly signed. These arguments specify the details of the recurrence relation for the generator:

```
x(i + 1) := (a * x(i) + c) mod m
```

Every discussion that I've seen of this algorithm uses unsigned values. Further, In C and C++ there is no modulus operator. The result of the `%` operator is implementation specific when either of its operands is negative, so implementing `mod` when the values involved can be negative requires a test and possible adjustment:

```
IntType res = (a * x + c) % m;
```

```
if (res < 0)
    res += m;
```

If the three template arguments can't be negative the recurrence relation can be implemented directly:

```
x = (a * x + c) % m;
```

This makes the generator faster.

Resolution:
In clause 5.1.4.1 [tr.rand.eng.lcong] replace every occurrence of `IntType` with `UIntType` and change the first sentence after the definition of the template from:

> The template parameter `IntType` shall denote an integral type large enough to store values up to `(m-1)`.

to:

> The template parameter `UIntType` shall denote an unsigned integral type large enough to store values up to `(m-1)`.

4.21 linear_congruential::linear_congruential(In&, In) -- Garbled Requires Clause

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The **Requires** clause for the member template `template <class In> linear_congruential(In& first, In last)` got garbled in the translation to .pdf format.

Resolution:
Change the **Requires** clause for the member template `template <class In> linear_congruential(In& first, In last)` in 5.1.4.1 [tr.rand.eng.lcong] from:

> **Requires:** `c > 0 – *first ¿ 0–`

to:

> **Requires:** `c > 0 || *first > 0`

4.22 bernoulli_distribution Isn't Really a Template

Submitter: Pete Becker (see N1535)
Status: Voted into the TR

The text says that `bernoulli_distribution` is a template, parametrized on a type that is required to be a real type. Its `operator()` returns a `bool`, with the probability of returning true determined by the argument passed to the object's constructor. The only place where the type parameter is used is as the type of the argument to the constructor. What is the benefit from making this type user-selectable instead of, say, double?

Resolution:

In 5.1.7.2 [tr.rand.dist.bern], change the section heading to "Class bernoulli_distribution", remove `template <class RealType = double>` from the declaration of `bernoulli_distribtion`, change the declaration of the constructor from:

```
explicit bernoulli_distribution(const RealType& p = RealType(0.5));
```

to:

```
explicit bernoulli_distribution(double p = 0.5);
```

and change the header for the subclause describing the constructor from:

```
bernoulli_distribution(const RealType& p = RealType(0.5))
```

to:

```
bernoulli_distribution(double p = 0.5)
```

4.23 Streaming Underspecified

Submitter: Pete Becker (see N1535)
Status: Open

See N1535 for a full discussion. Summary: the goal is for engines to be well enough specified so that the state of an engine can be streamed out on one system and read in on a different system, and so that the engine on the second system would produce the same sequence of values as it would on the first. Distributions are less clear-cut, but at least we want to be able to save and restore on the same system for the sake of checkpointing. Given that we don't care about portability, streaming of distributions may be adequately specified. However, we may not want to call it `operator<<` and `operator>>`, because implementers will probably want to use binary formats.

4.24 Garbled characters

Submitter: Jens Maurer
Status: Editorial

There are some places where the TR draft contains garbled characters. This issue points out the places where editorial changes to rectify this need to be performed.

- 5.1.4.3 [tr.rand.eng.sub], first paragraph
- 5.1.4.4 [tr.rand.eng.sub1], first paragraph
- 5.1.4.5 [tr.rand.eng.disc], after the class definition
- 5.1.4.5 [tr.rand.eng.disc], effects clause of operator()

4.25 class vs. type

Submitter: Jens Maurer
Status: Voted into the TR

The wording in section 5.1.1 isn't parallel.

Resolution: Replace in section 5.1.1 [tr.rand.req], last paragraph
In the following subclauses, a template parameter named UniformRandomNumberGenerator shall denote a class **type** that satisfies all the requirements of a uniform random number generator.

4.26 Fix section reference

Submitter: Jens Maurer
Status: Voted into the TR, Editorial

A section reference needs to be fixed.

Resolution:
Replace in section 5.1.4 [tr.rand.eng], second paragraph

> The class templates specified in this section satisfy all the requirements of a pseudo-random number engine (given in tables in section ~~x.x.x~~ **5.1.1 [tr.rand.req]**), except where specified otherwise. Descriptions are provided here only for operations on the engines that are not described in one of these tables or for operations where there is additional semantic information.

4.27 Avoid confusion for "ell" and "one"

Submitter: Jens Maurer
Status: Voted into the TR

We need to be careful with subscripts: “l” and “1” look very similar in most fonts, so “l” is a poor choice for a variable that will be used in subscripts.

Resolution:
Replace in 5.4.1.2 [tr.rand.eng.mers]

> Effects: With a linear congruential generator l(i) having parameters m_l = 232, al = 69069, c_l = 0, and l(0) = value, sets x(-n) ... x(-1) to l(1) ... l(n), respectively.

by

> Effects: With a linear congruential generator lcg(i) having parameters m_{lcg} = 232, a_{lcg} = 69069, c_{lcg} = 0, and lcg(0) = value, sets x(-n) ... x(-1) to lcg(1) ... lcg(n), respectively.

Replace in 5.4.1.3 [tr.rand.eng.sub]

> **Effects:** With a linear congruential generator l(i) having parameters m_l = 2147483563, a_l = 40014, c_l = 0, and l(0) = `value`, sets x(-r) ... x(-1) to l(1) mod m ... l(r) mod m, respectively. If x(-1) == 0, sets carry(-1) = 1, else sets carry(-1) = 0.

by

> **Effects:** With a linear congruential generator lcg(i) having parameters m_{lcg} = 2147483563, a_{lcg}= 40014, c_{lcg} = 0, and lcg(0) = `value`, sets x(-r) ... x(-1) to lcg(1) mod m ... lcg(r) mod m, respectively. If x(-1) == 0, sets carry(-1) = 1, else sets carry(-1) = 0.

Replace in 5.4.1.4 [tr.rand.eng.sub1]

> **Effects:** With a linear congruential generator l(i) having parameters m = 2147483563, a = 40014, c = 0, and l(0) = `value`, sets x(-r) ... x(-1) to (l(1)*2^{-w}) mod 1 ... (l(r)*2^{-w}) mod 1, respectively. If x(-1) == 0, sets carry(-1) = 2^{-w}, else sets carry(-1) = 0.

by

> **Effects:** With a linear congruential generator lcg(i) having parameters m_{lcg} = 2147483563, a_{lcg}= 40014, c_{lcg} = 0, and lcg(0) = `value`, sets x(-r) ... x(-1) to (lcg(1)*2^{-w}) mod 1 ... (lcg(r)*2^{-w}) mod 1, respectively. If x(-1) == 0, sets carry(-1) = 2^{-w}, else sets carry(-1) = 0.

[Note to editor: see issue 19 for another issue that touches these words.]

4.28 xor_combine: fix typo

Submitter: Jens Maurer
Status: Voted into the TR

Resolution:
Replace in 5.1.4.6 [tr.rand.eng.xor]

> The template parameters UniformRandomNumberGenerator1 and UniformRandomNumberGenerator~~1~~**2** shall denote classes that satisfy all the requirements of a uniform random number generator, ...

[Replace "1" by "2" once.]

4.29 Require additional properties for Engine result_type

Submitter: Jens Maurer
Status: Voted into the TR

Currently, there are no restrictions on `UniformRandomNumberGenerator::result_type`, although`variate_generator` is supposed to possibly convert between integer and floating-point types.

Proposed resolution:
In 5.1.1 [tr.rand.req], replace the pre/post-condition for `result_type`:

> `std::numeric_limits<T>::is_specialized` is `true`

by

> `T` is an arithmetic type [basic.fundamental]

4.30 Garbled precondition for min()

Submitter: Jens Maurer
Status: Voted into the TR

Proposed resolution:
In 5.1.3 [tr.rand.var], add the highlighted text for `min()`:

> Precondition: distribution().min() is **well-formed**

4.31 xor_combine: Require additional properties for base*_type::result_type

Submitter: Jens Maurer
Status: Voted into the TR

There are no restrictions on UniformRandomNumberGenerator1::result_type and UniformRandomNumberGenerator2::result_type that would ensure that << and ^ are available on them. That's well defined for unsigned integral types.

Proposed resolution:
Add in 5.1.4.6 [tr.rand.eng.xor] in the paragraph after the class definition

Both UniformRandomNumberGenerator1::result_type andUniformRandomNumberGenerator2::result_type shall denote (possibly different) unsigned integral types. The size of the state ...

4.32 Be precise about the size of the state of xor_combine

Submitter: Jens Maurer
Status: Voted into the TR

It is unclear what the "size of b1" and the "size of b2" mean, we only talk about the "size of the state".

Proposed resolution:
Add in 5.1.4.6 [tr.rand.eng.xor] in the paragraph after the class definition:
The size of the state is the size **of the state** of b1 plus the size **of the state** of b2.

4.33 uniform_real should return open interval

Submitter: Jens Maurer
Status: Voted into the TR

uniform_real was specified with a closed interval [min, max] range, but it should have a half-open interval [min, max) range to avoid lots of special cases in more complex distributions. (The boost implementation and documentation does this since ever.)

Proposed resolution:

In 5.1.7.6 [tr.rand.dist.runif], replace
min <= x <= max
by
min <= x < max

4.34 No complexity specification for copy construction and copy assignment

Submitter: Jens Maurer
Status: Voted into the TR

In 5.1.1 [tr.rand.req], add a new paragraph after table 5.3 (pseudo-random number generator):
Additional requirements: The complexity of both copy construction and assignment is O(size of state).

4.35 Insufficient preconditions on discard_block

Submitter: Jens Maurer
Status: Voted into the TR

discard_block does not have sufficient requirements on the r and p template parameters.

Proposed resolution:

Replace in 5.1.4.5 [tr.rand.eng.disc]
  r <= q
by
  The following relation shall hold: 0 <= r <= p.

4.36 Insufficient preconditions on xor_combine

Submitter: Jens Maurer
Status: Voted into the TR

xor_combine does not have any requirements for s1 and s2 template parameters.

Proposed resolution:

Add in 5.1.4.6 [tr.rand.eng.xor], paragraph after the class definition, before "The size of the state ..."
  The following relation shall hold: 0 <= s1 and 0 <= s2.

5 Special function issues

5.1 Clean up special function names and descriptions

Submitter: Bill Plauger, Walter Brown
Status: Voted into the TR

The names of special functions should be cleaned up so they're all-lowercase and more spelled out (to make them more consistent with C naming style), there should be names with ***f*** and ***l*** suffixes for float and long double versions, and the behavior should be specified mathematically instead of by reference.

Resolution:
Accept the changes proposed in N1542, "Mathematical special functions, v3".

6 Unordered associative container issues

6.1 Incorrect const qualification

Submitter: Rober Klarer
Status: Voted into the TR

The parameters to the container swap functions are const-qualified, and I don't think they should be. For example the declaration for the swap function that appears in 6.2.4.3.2 is

```
template <class Value, class Hash, class Pred, class Alloc>
void swap(const unordered_set<Value, Hash, Pred, Alloc>& x,
          const unordered_set<Value, Hash, Pred, Alloc>& y);
```

I believe that x and y can't be references to const containers because the swap function needs to be able to modify both containers.

Resolution:
In section 6.4.2 [tr.unord.unord], remove the const qualification in the parameters of the nonmember swap functions for all four unordered associative containers, both in the header synopses and in the text.

6.2 Erase takes const iterator

Submitter: Rober Klarer
Status: NAD

The erase member functions with iterator parameters are declared as follows

```
  void erase(const_iterator position);
  void erase(const_iterator first, const_iterator last);
```

This is consistent with the requirements table, but I'm not sure that it's intentional.

Resolution: Not a defect. This was intentional. The other containers should probably be changed in a similar way in a future standard.

6.3 Bucket members not declared const

Submitter: Rober Klarer
Status: Voted into the TR

The bucket(...) and bucket_size(...) members of each container template should be const, but they aren't declared const in the class definitions. The requirements table correctly implies that these functions are const members.

Resolution:
In section 6.4.2 [tr.unord.unord], in the class declarations of all four unordered associative containers, declare the bucket and bucket_size member functions as const.

6.4 Incorrect variable in requirements table

Submitter: Rober Klarer
Status: Voted into the TR

All occurences of "for const a" in the "Return Type" column of the requirements table should actually read "for const b." Also, under the the "assertion/note/pre/postcondition" column, the phrase "out of which a was constructed" should be "out of which b was constructed" for b.hash_function() and b.key_eq(). Similarly, "a.end()" should be "b.end" for b.find(k), and "std::make_pair(a.end(), a.end())" should be "std::make_pair(b.end(), b.end())" for b.equal_range(k).

Resolution:
As above. (See N1549.)

7 Regular expression issues

7.1 *basic_regex* should Not Keep a Copy of its Initializer

Submitter: Pete Becker (N1499)
Status: Voted into the TR

The `basic_regex` template has a member function `str` which returns a string object that holds the text used to initialize the `basic_regex` object. It also provides a container-like interface to this text through the member functions`begin` and `end`, which return `const_iterator` objects that allow inspection of the initializer text. While it might occasionally be useful to look at the initializer string, we ought to apply the rule that you don't pay for it if you don't use it. Just as `fstream` objects don't carry around the file name that they were opened with, `basic_regex` objects should not carry around their initializer text. If someone needs to keep track of that text they can write a class that holds the text and the `basic_regex` object.

Resolution:
As described in N1551, Changes to N1540 to Implement N1499 Parts 1 and 2.

7.2 basic_regex Should Not Have an Allocator

Submitter: Pete Becker (N1499)
Status: Voted into the TR

The `basic_regex` template takes an argument that defines a type for an allocator object. The template also has several member typedefs and one member function to provide information about the allocator type and the allocator object. This is because a `basic_regex` object "is in effect both a container of characters, and a container of states, as such an allocator parameter is appropriate." Calling it a container doesn't make it one. The allocator in `basic_regex` is not very useful, and it unduly complicates the implementation.

The cost of using an allocator is high. Every type that the `basic_regex` object uses internally must have its own allocator type and its own allocator object. A node based implementation might have a dozen or more node types, requiring a dozen or more allocator objects. Allocator objects can be created as local objects when needed, which effectively precludes allocators with internal state; they can be ordinary members of the `basic_regex` object, inflating its size; or they can be implemented as a chain of base classes (to take advantage of the zero-size base optimization), with a high cost in readability and maintainability. None of these options is attractive.

Further, it's not at all clear how a user can determine that a substitute allocator is appopriate or what characteristics such an allocator should have. The STL containers have clearly spelled out requirements for their memory usage;`basic_regex` objects have no such requirements (nor should they). The implementor of the `basic_regex` template knows best what its memory requirements are.

Resolution:

As described in N1551, Changes to N1540 to Implement N1499 Parts 1 and 2. Some memory management interface may be a good idea, but allocators aren't it.

7.3 The Interface to regex_traits Should Use Iterators, Not Strings

Submitter: Pete Becker (N1499)
Status: Open

The member functions of the `regex_trait` template support customization and internationalization for regular expressions. Of these, the member functions `transform`, `transform_primary`, `lookup_collatename`, and`lookup_classname` take `string` as input.

This interface is inherently inefficient -- it requires creating a string object from a sequence in order to pass that string to the function. Further, in the case of `transform`, the function typically extracts iterators from the string object. Passing the text as a pair of iterators avoids introducing unnecessary string objects.

Resolution:
The LWG thought this seemed like a good idea, but the details need to be worked out. Note that the iterators need to be ForwardIterator, not InputIterator.

7.4 Regular expressions and internationalization

Submitter: Pete Becker (N1500)
Status: Open

See N1500 for a detailed description. Summary: We're basing regexps on ECMAScript. However, ECMAScript is entirely unicode and doesn't deal with multiple locales and such. We're using it in a non-unicode environment. Some of the lookups it's asking for, e.g. asking whether a character is a digit in a locale-dependent way, are very expensive.

We allow metacharacters to be remapped, and (via the *translate* member function) even ordinary characters may be remapped. Remapping metacharacters means you can't tell what a regexp does just be looking at it. Remapping ordinary characters means that we use an expensive code path for all matches, even ordinary case sensitive matches.

Suggestions:

- Don't use translate for case-sensitive matches. (Or at least only use it if we're using the *collate* option when compiling the regex string into the regex object.
- Get rid of the *syntax_type* function that allows you to remap the meaning of metacharacters.

Resolution:
Discussed at Kona, the LWG was generally sympathetic to this simplification. The one Japanese representative in the room thought that this was a good idea, and that it matches the way that Japenese programmers use regular expressions. The LWG believes we should make these changes at the next meeting, pending specific wording.

7.5 Bad rationale for regex_ prefixes

Submitter: Pete Becker (N1507)
Status: NAD

Pete writes:

> I'm not strongly for or against the regex_ prefixes. They may well be helpful in understanding code. But I'm strongly against the notion that the standard library should use prefixes because users abuse using declarations.

Resolution: NAD. The rationale isn't part of the TR. If we decide to change the names, that will be a separate issue.

7.6 Unintended occurrence of reg_expression

Submitter: John Maddock (N1507)
Status: Voted into the TR

There is a systematic error in the "proposed text" section: the various algorithms have been defined to accept a type "reg_expression" which does not in fact exist in the proposal, and which should of course be called "basic_regex". This is an editing error that crept in when the name of that class was changed from reg_expression to basic_regex.

The fix is to just replace all occurrences of "reg_expression" with "basic_regex" throughout that section.

Resolution: As above.

7.7 Iterators have incorrect definitions of the types "reference" and "pointer"

Submitter: John Maddock (N1507)
Status: Voted into the TR

In regex_iterator and regex_token_iterator the definitions given for the types "iterator" and "reference" are wrong: as given these types refer/point to the value_type of the underlying iterator type, but should of course refer/point to the actual value_type being enumerated (the two are not the same type).

Resolution:
Change:

```
typedef typename
iterator_traits<BidirectionalIterator>::pointer
        pointer;
typedef typename
iterator_traits<BidirectionalIterator>::reference
        reference;
```

To:

```
typedef const value_type* pointer;
typedef const value_type& reference;
```

In both the regex_iterator and regex_token_iterator definitions.

7.8 regex_iterator does not handle zero-length matches correctly

Submitter: John Maddock (N1507)
Status: Open

There is a subtle bug in regex_iterator::operator++; when the previous match found matched a zero-length string, then the iterator needs to take special action to avoid going into an infinite loop, the current wording does this but gets it wrong because it does not allow two consecutive zero length matches, for example iterating occurrences of "^" in the text "\n\n" yields just one match rather than three as it should. The actual behavior should be as follows:

When the previous match was of zero length, then check to see if there is a non-zero-length match starting at the same position, otherwise move one position to the right of the last match (if such a position exists), and continue searching as normal for a (possibly zero length) match.

Resolution:
Discussed at Kona. Leaving open because this is tied up with the next issue, which we need better wording for.

7.9 Regex_iterator does not set match_results::postion correctly

Submitter: John Maddock (N1507)
Status: Open

As currently specified, given:

```
regex_iterator<something> i;
```

then `i->position() == i->prefix().length()` for all matches found.

This is correct for the first match found, but makes little sense for subsequent matches where the result of i->position() is only useful if it returns the distance from the start of the string being searched to the start of the match found.

(Recall that i->prefix() contains everything from the end of the last match found, to the start of the current match, this allows search and replace operations to be constructed by copying i->prefix() unchanged to output, and then outputting a modified version of whatever matched.)

For example this problem showed up when converting a boost.regex example program from the regex_grep algorithm (not part of the proposal) to use regex_iterator: the example takes the contents of a C++ source file as a string, and creates an index that maps C++ class names to file positions in the form of a std::map<std::string, int>. In order for the program to take a regex_iterator and from that add an item to the index, it needs to know how far it is from the start of the text being searched to the start of the current match: that was what regex_match::position() was intended for, but as the proposal stands it instead returns the distance from the end of the last match to the start of the current match.

Resolution:
Discussed at Kona. General agreement that this is a real issue, also that the proposed resolution in N1507 is not the right way to resolve it. The proposed resolution replaces a bunch of code

with another bunch of even more complicated code. (As we found in the I/O clauses, the trouble with specifying behavior in terms of code is that code can be buggy.) We need to describe the behavior in text, not as a code fragment or a table.

7.10 Naming of basic_regex::getflags

Submitter: Pete Becker (N1507)
Status: Voted into the TR

basic_regex has member functions named getflags and get_allocator. The latter is consistent with the use of the same name in STL containers. In general, it seems to me, the library tries to use an underscore to separate a verb from its object for names of this nature. That convention would mean that we should call the other one get_flags. On the other hand, we do have getline, but that's arguably different because it's not a state query. Do we have a general policy here? If so, what is it, and what should the name of getflags be?

Resolution:
Replace all occurrences of "getflags" in the document with "flags".

7.11 Missing namespace prefix in regex_iterator description

Submitter: Pete Becker (N1507)
Status: Voted into the TR

The definition of regex_iterator in RE.8.1 mentions
regex_iterator(BidirectionalIterator a, BidirectionalIterator b, const regex_type& re,
match_flag_type m = match_default);

And
match_flag_type flags; // for exposition only

match_flag_type and match_default are defined in the nested namespace regex_constants, so these two names need to be qualified with regex_constants::. Same thing in the first RE.8.1.1.

Resolution:
Go through the text and replace all occurrences of:
`match_flag_type` with `regex_constants::match_flag_type`,
`match_default` with `regex_constants::match_default`,
`match_partial` with `regex_constants::match_partial`,
`match_prev_avail` with `regex_constants::match_prev_avail`,
`match_not_null` with `regex_constants::match_not_null`,
`format_default` with `regex_constants::format_default`,
`format_no_copy` with `regex_constants::format_no_copy`,
`format_first_only` with `regex_constants::format_first_only`,
except in the section which defines these (RE.3.1).

7.12 Unnecessary sub-section headers in regex_iterator

Submitter: Pete Becker (N1507)
Status: editorial, voted into the TR

The first clause labeled RE.8.1.1 has the title "regex_iterator constructors". It contains descriptions of the constructors, plus several operators. The second clause labeled RE.8.1.1 has the title "regex_iterator dereference". It contains operator*, operator->, and the two versions of operator++. Seems like both of these labels should be removed.

Resolution:
Rename the section “RE.8.1.1 regex_iterator constructors” as “regex_iterator members”, remove the section “RE.8.1.1 regex_iterator dereference”, rename the section “RE.8.2.1 regex_iterator constructors” as “regex_token_iterator members”, remove the section: “RE.8.2.1 regex_token_iterator dereference”.

7.13 Names of symbolic constants

Submitter: Pete Becker (N1507)
Status: voted into the TR

ECMAScript has five control escapes: t, n, v, f, r. The regex proposal has named constants for four of them: escape_type_control_f, _n, _r, and _t. escape_type_control_v seems to be missing. (Okay, that's not about names, but the next two are).

This is minor, but in C and C++ those five things are escape sequences, and using names that include 'control' is a bit confusing. Granted, it fits with the terminology in ECMAScript, but I'd lean toward more C-like names, on the line ofescape_type_f.

And finally, there's escape_type_ascii_control. (For those not familiar with the details of the proposal, this refers to things that we might write in ordinary text as <ctrl>-X, for example.) We've pretty much avoided the term "ascii" in the standard (it's only used twice, in footnotes, apologetically), and I'm a bit uncomfortable with its use here. I'd preferescape_type_control_letter, which picks up the name of the production in the ECMAScript grammar for the letter that follows the escape. I think it's pretty clear what it means, and it avoids "ascii".

Resolution:
Replace all occurrences of:
    escape_type_control_f with escape_type_f
    escape_type_control_n with escape_type_n
    escape_type_control_r with escape_type_r
    escape_type_control_t with escape_type_t
    escape_type_ascii_control with escape_type_control

Then immediately after the line:
    static const escape_syntax_type escape_type_t;
add the line:
    static const escape_syntax_type escape_type_v;

Then immediately after the table entry:
        escape_type_t   t

Add the new table entry:

escape_type_v	v

[Kona: in addition to the proposed resolution in this issue: the LWG felt that a review of names throughout the regex clause is in order: the names tend to be verbose. See issue 7.41.]

7.14 Traits class versioning incompletely edited in.

Submitter: Pete Becker (N1507)
Status: Open

The paper talks about versioning of regex_traits classes, and RE.1.1 (in table RE2) says that a traits class shall have a member X::version_tag whose type is regex_traits_version_1_tag or a class that publicly inherits from that. Neither the <regex> synopsis (RE.2) nor the description of regex_traits (RE.3.3) mentions either of these types. I can't tell whether this was partially edited in or partially edited out. <g> So, is regex_traits versioning part of the proposal?

Resolution:
Discussed at Kona. It's unclear whether this solves the versioning problem and it's also unclear whether an ad hoc solution that applies only to regular expressions, instead of a solution to the versioning problem in general, is a good idea.

7.15 Specification of sub_match::length incorrect

Submitter: John Maddock (N1507)
Status: voted into the TR

The specification for sub_match::length has acquired a couple of typos (a misplaced static, and the logic in the effects clause is back-to-front)

Resolution:
Change it to:

```
difference_type length();
```

Effects: returns `(matched ? distance(first, second) : 0)`.

*[Note to editor: throughout the regex section, we see "**Effects:** returns…" This is unnecessarily convoluted, and should be replaced with plan "**Returns:** …"]*

7.16 Traits class sentry language

Submitter: Pete Becker (N1507)
Status: Open

The proposal says:

> "An object of type regex_traits<charT>::sentry shall be constructed from a regex_traits object, and tested to be not equal to null, before any of the member functions of that object other than `length`, `getloc`, and `imbue` shall be called. Type sentry performs implementation defined initialization of the traits class object, and represents an opportunity for the traits class to cache data obtained from the locale object."

The first sentence is in passive voice, and begs the question of who shall do it: the user of the regex instance that holds the regex_traits object, or the regex instance itself. Unless the user is hacking around with a standalone instance ofregex_traits, it probably ought to be the regex object that "shall" do this.

Second, sentry "performs implementation defined initialization." I think this ought to be implementation specific, not implementation defined. I don't want to have to document the details of the initialization that sentry performs.

Resolution:
Agreed that this is a real problem. However, the wording proposed in N1507 fails to solve the problem of clarifying whether it's the implementation or the user who constructs sentries.

7.17 Imprecise specification of regex_traits::char_class_type

Submitter: Pete Becker (N1507)
Status: voted into the TR

Roughly speaking, there are three categories of character class: the ones that are supported by C and C++ locales (alnum, etc.), the additional ones for the regex proposal (d s w) and user-supplied character classes (through extensions to regex_traits).

Is the intent of the proposal to require that for the first category, the value returned by, for example,lookup_classname("alnum") be the value alnum as defined by ctype_base::mask? (I don't care one way or the other, but we have to be clear about what's required).

Resolution:
Replace:

> "The type `char_class_type` is used to represent a character classification and is capable of holding an implementation defined superset of the values held by ctype_base::mask (22.2.1)."

with:

> "The type `char_class_type` is used to represent a character classification and is capable of holding the implementation specific set of values returned by `lookup_classname`."

7.18 Can anything other than basic_regex throw bad_expression objects?

Submitter: Pete Becker (N1507)
Status: Open

The text describing the class bad_expresions says it is the type of the object thrown to report errors "during the conversion from a string ... to a finite state machine." This suggests that it is not thrown by the functions that try to match a string to and a basic_regex object, and this is borne out by the throws clauses for the constructors and assignment operators for basic_regex, which say that they throw bad_expression if the string isn't a valid regular expression, and by the lack of throws clauses for regex_match, etc.

On the other hand, error_type has two values, error_complexity and error_stack, that only occur

during matching. There's no other mention of these values, so the only thing that can be done with them is for the implementation to pass them to regex_traits::error_string, and the only way the user can see the resulting string is by catching an exception. This suggests that bad_expression can also be thrown by the match functions. And the text says, in the last paragraph of RE.4, that "the functions described in this clause can report errors by throwing exceptions of type bad_expression."

So: can the various match functions throw bad_expression, and, if so, is bad_expression the appropriate name?

Resolution:
Discussed at Kona. This is a real problem. However, the wording proposed in N1507 doesn't solve the problem. The extra flags Pete notices are still there, and so is the problematic sentence about seemingly inappropriate functions being able to throw exceptions. Finally, the LWG wants to know the actual type of the thrown exception object.

7.19 Unneeded basic_regex members

Submitter: John Maddock
Status: voted into the TR

The following basic_regex members are redundant and should be removed:

```
basic_regex(const charT* p1, const charT* p2, flag_type f =
regex_constants::normal,
                    const Allocator& a = Allocator());
basic_regex& assign(const charT* first, const charT* last,
                                    flag_type f =
regex_constants::normal);
```

Resolution: As above.

7.20 Missing basic_regex members

Submitter: Pete Becker (N1507)
Status: voted into the TR

The proposal has member functions named 'assign' that take argument lists that correspond to the argument lists for constructors, with two exceptions: there's `basic_regex(const charT *, size_type len,flag_type)`, but no `assign(const charT *, size_type, flag_type)`; and there's `basic_regex()`, but no`assign()`. Are these omissions intentional?

Resolution:
add the following member to the basic_regex class synopsis:

basic_regex& assign(const charT* ptr, size_type len, flag_type f = regex_constants::normal);

Then add the following description in the RE4.5 section:

basic_regex& assign(const charT* ptr, size_type len, flag_type f = regex_constants::normal);

Effects: Returns assign(string_type(ptr, len), f).

7.21 Types of match_results typedefs members

Submitter: Pete Becker (N1507)
Status: voted into the TR

The proposal says that match_results has a nested typedef
```
typedef const value_type& const_reference
```

Since match_results has an allocator, this should be
```
typedef typename allocator::const_reference const_reference

Resolution: As above
```

7.22 What does match_results::size() return?

Submitter: Pete Becker (N1507)
Status: voted into the TR

The member funtion size() returns "the number of sub_match elements stored in *this". Aside from the suggested implementation above, there are the prefix() and suffix() sub_match elements. Is the intention that size() should return the number of capture groups in the original expression, and not include those two extra sub_matches? (I think the answer is probably yes).

Resolution:
Replace:
```
size_type size()const;
```

Effects: Returns the number of sub_match elements stored in *this.

With:
```
size_type size()const;
```

Effects: Returns one plus the number marked sub-expressions in the regular expression that was matched.
[Note to editor: put in the missing "of"]

7.23 What does match_results::position return when passed an out of range index?

Submitter: Pete Becker
Status: voted into the TR

match_results::position() doesn't say what happens when someone asks for the position of a non-matched group. The specification says that it's distance(first1, first2), where first1 is the beginning of the target text and first2 is the beginning of the nth match. The specification for sub_match says that for a failed match the iterators have unspecified contents. Do we want this to be unspecified or undefined, or is there some meaningful value we can return?

Having looked ahead <g>, the match and search algorithms specify that non-matched groups hold iterators that point to the end of the target text. This conflicts with the specification for sub_match, which says they're undefined. Is that text in sub_match incorrect?

Resolution:
Changes to:

> `difference_type position(unsigned int sub = 0) const;`
> **Effects:** Returns `std::distance(prefix().first, (*this)[sub].first)`.

Are covered in "Regex_iterator does not set match_results::postion correctly".

Delete the following paragraphs from the sub_match specification:

> When the marked sub-expression denoted by an object of type sub_match<> participated in a regular expression match then member `matched` evaluates to true, and members `first` and `second` denote the range of characters `[first,second)` which formed that match. Otherwise `matched` is false, and members `first` and`second` contained undefined values.
>
> If an object of type `sub_match<>` represents sub-expression 0 - that is to say the whole match - then member`matched` is always true, unless a partial match was obtained as a result of the flag `match_partial` being passed to a regular expression algorithm, in which case member `matched` is false, and members `first` and `second`represent the character range that formed the partial match.

The add the following to the match_results specification, immediately after the sentence ending *"except that only operations defined for const-qualified Sequences are supported."*:

> The `sub_match<>` object stored at index zero represents sub-expression 0; that is to say the whole match. In this case the `sub_match<>` member `matched` is always true, unless a partial match was obtained as a result of the flag `regex_constants::match_partial` being passed to a regular expression algorithm, in which case member `matched` is false, and members `first` and `second` represent the character range that formed the partial match.
>
> The `sub_match<>` object stored at index *n* denotes what matched the marked sub-expression *n* within the matched expression. If the sub-expression *n* participated in a regular expression match then the `sub_match<>` member `matched` evaluates to true, and members `first` and `second` denote the range of characters `[first,second)` which formed that match. Otherwise `matched` is false, and members `first` and `second` point to the end of sequence that was searched.

7.24 What happens if match_results::operator[] is out of range?

Submitter: Pete Becker
Status: voted into the TR

With respect to match_results::operator[]: We need to say what happens for an index out of range. Seems to me there are two reasonable possibilities: undefined behavior, or returns a no-match object.

While I strongly favor undefined behavior over artificially well-defined results, I also favor well-

defined behavior when it is not too artificial. Thus, the behavior of sqrt(-2.0) is undefined; free(0) does nothing. While undefined behavior provides a convenient hook for debugging implementations, that's not its purpose, and if we can specify reasonable (which includes inexpensive) behavior we ought to do it, rather than provide another place where users can go astray.

In this case, I think I prefer to view operator[] as indexing into an unbounded array of sub_match objects. The objects at match_results.size() and above would look like failed sub-matches: their boolean flag would be false, and both their iterators would point to the end of the target string. Since we've agreed that sub_match objects for failed sub-matches need not have distinct addresses, this can be implemented by simply adding one sub_match element beyond those needed for the actual results, and returning it for an index that's otherwise out of bounds.

Resolution:
replace:

```
const_reference operator[](int n) const;
```

Effects: Returns a reference to the `sub_match` object representing the character sequence that matched marked sub-expression *n*. If `n == 0` then returns a reference to a `sub_match` object representing the character sequence that matched the whole regular expression.

With:

```
const_reference operator[](int n) const;
```

Effects: Returns a reference to the `sub_match` object representing the character sequence that matched marked sub-expression *n*. If `n == 0` then returns a reference to a `sub_match` object representing the character sequence that matched the whole regular expression. If n >= size() then returns a sub_match object representing an unmatched sub-expression.

7.25 Incorrect case insensitive match specification

Submitter: John Maddock (N1507)
Status: closed

The following wording:

> "During matching of a regular expression finite state machine against a sequence of characters, comparison of a collating element range c1-c2 against a character c is conducted as follows: if getflags() &regex_constants::collate is true, then the character c is matched if traits_inst.transform(string_type(1,c1)) <= traits_inst.transform(string_type(1,c)) && traits_inst.transform(string_type(1,c)) <= traits_inst.transform(string_type(1,c2)), otherwise c is matched if c1 <= c && c <= c2. During matching of a regular expression finite state machine against a sequence of characters, testing whether a collating element is a member of a primary equivalence class is conducted by first converting the collating element and the equivalence class to a sort keys using traits::transform_primary, and then comparing the sort keys for equality."

Is defective in that it does not take account of case-insensitive matches, all input characters, and all collating elements in the finite state machine should be passed through traits.inst.translate before being converted into a sort key.

Resolution: Closed, this is covered by the issue 7.26.

7.26 Character class extensions to ECMAScript grammar need a formal grammar

Submitter: Pete Becker (N1507)
Status: voted into the TR

The regex proposal adds to ECMAScript the ability to use named character classes through "expressions of the form":

[[:class-name:]]
[[.collating-name.]]
[[=collating-name=]]

This isn't sufficient. In ECMAScript the expression [[] is valid, and names a character set containing the character '['. Similarly, [[:] is also valid, and names a character set containing the characters '[' and ':'. We need to say whether these two expressions (and their analogs for collating names) are still valid. I suspect the answer is that they're not -- a '[' as the first character in a character class is a special character, which must be follwed by one of ':', '.', or '=', then a name that does not contain any of ']', ':', ".', or '=' (technically we could allow ']', but that seems unnecessarily baroque), then the appropriate close marker.

Resolution: Adopt the proposed resolution in N1507.

7.27 Imprecise Specification of regex_replace

Submitter: Pete Becker (N1507)
Status: voted into the TR

Finds all the non-overlapping matches 'm' of type match_results<BidirectionalIterator> that occur in the sequence [first, last).

Having found them or not, it then writes stuff depending on its arguments. It's not clear, though, what "non-overlapping matches" are. It took me about five minutes to convince myself that these are matches of the complete expression, and not matches of internal capture groups (which would always overlap the full match). I think a footnote is sufficient for this. More important, though, is what happens when matches overlap. Suppose we're searching for "aba" in the text "ababa". There are two matches: the first three characters match, and the last three match. These two matches overlap. Do we discard them both? Keep the first? Keep the second? My guess is that the intention is to keep the first one, but we need to say so.

Resolution:
Replace the following clause:

Effects: Finds all the non-overlapping matches *m* of type `match_results<BidirectionalIterator>` that occur within the sequence [first, last). If no such matches are found and `!(flags & format_no_copy)` then calls

`std::copy(first, last, out)`. Otherwise, for each match found, if `!(flags & format_no_copy)` calls `std::copy(m.prefix().first, m.prefix().last, out)`, and then calls `m.format(out, fmt, flags)`. Finally if `!(flags & format_no_copy)` calls `std::copy(last_m.suffix().first,last_m,suffix().last, out)` where `last_m` is a copy of the last match found. If `flags &format_first_only` is non-zero then only the first match found is replaced.

With:

Effects: Constructs an regex_iterator object: `regex_iterator<BidirectionalIterator, charT, traits, Allocator> i(first, last, e, flags)`, and uses *i* to enumerate through all of the matches *m* of type`match_results<BidirectionalIterator>` that occur within the sequence [first, last). If no such matches are found and `!(flags & format_no_copy)` then calls `std::copy(first, last, out)`. Otherwise, for each match found, if `!(flags & format_no_copy)` calls `std::copy(m.prefix().first, m.prefix().last, out)`, and then calls `m.format(out, fmt, flags)`. Finally if `!(flags & format_no_copy)` calls`std::copy(last_m.suffix().first, last_m,suffix().last, out)` where `last_m` is a copy of the last match found. If `flags & format_first_only` is non-zero then only the first match found is replaced.

7.28 What is an invalid/empty regular expression?

Submitter: Pete Becker (N1507)
Status: Open

See N1507 for a full description. Summary: it's not clear what kind or regex object the default constructor returns, and how that interacts with the `empty()` test.

Resolution:
Discussed at Kona. The LWG agrees that the default constructor should be equivalent to construction from an empty string. Leaving this open for now partly because we need wording expressing that, and partly because it's not clear that there's any point to having the `empty()` member function in the first place.

7.29 Regular expression constructor language

Submitter: Pete Becker (N1507)
Status: Open

For the basic_regex ctor that takes a const charT *p, the proposal says:
Effects: Constructs an object of class basic_regex; the object's internal finite state machine is constructed from the regular expression contained in the null-terminated string p...

p is not a null-terminated string. It is a pointer. The analogous phrasing for basic_string is:
Effects: Constructs an object of class basic_string and determines its initial string value from the array of charTof length traits::length(s) whose first element is designated by s ...

We need to maintain a similar level of formalism.

Resolution:
The LWG agrees the current wording is too imprecise. Leaving this open until we get the formalized wording this issue calls for.

7.30 Incorrect usage of "undefined"

Submitter: Pete Becker (N1507)
Status: Voted into the TR

In several places in the document the term "undefined" should be replaced by "unspecified":

"Otherwise `matched` is false, and members `first` and `second` contained ***undefined*** values."

"If the function returns false, then the effect on parameter *m* is ***undefined***, otherwise the effects on parameter *m* are given in table RE18"

"If the function returns false, then the effect on parameter *m* is ***undefined***, otherwise the effects on parameter *m* are given in table RE19"

Resolution: As above

7.31 Incorrect usage of "implementation defined"

Submitter: Pete Becker (N1507)
Status: Voted into the TR

In several places in the document the term "implementation defined" should be replaced by either "implementation specific" or "unspecified":

"Type sentry performs ***implementation defined*** initialization of the traits class object, and represents an opportunity for the traits class to cache data obtained from the locale object."
`"char_class_type lookup_classname(const string_type& name) const;`

Effects: returns an ***implementation defined*** value that represents the character classification `name"`

"Returns: converts f into a value `m` of type `ctype_base::mask` in an ***implementation defined*** manner"

"Effects: constructs an object `result` of type `int`. If `first == last` or if `is_class(*first,lookup_classname("d")) == false` then sets `result` equal to -1. Otherwise constructs a `basic_istream<charT>`object which uses an ***implementation defined*** stream buffer type which represents the character sequence [first,last), and sets the format flags on that object as appropriate for argument `radix`."

Resolution: As above

7.32 Are sub_match objects all unique?

Submitter: Pete Becker (N1507)
Status: NAD

Are sub_match objects for non-matched capture groups required to be distinct? I can picture amatch_type implementation that holds sub_match objects only for the capture groups that matched, and returns a generic no-match object for others. Is this intended to be legal? (My inclination is that it ought to be allowed, because I don't see any good reason not to allow it).

Resolution:
No, match objects are not guaranteed to be unique; the lack of a guarantee was intentional.
[Editorial issue: The editor should add a non-normative note pointing that out.]

7.33 How are Unicode escape sequences handled?

Submitter: Pete Becker (N1507)
Status: Open

ECMA-Script supports character escapes of the form "\uxxxx", where each 'x' is a hex digit. Each such escape sequence represents the character whose code point is the value of 'xxxx' translated to a number in the usual way. What do such character escapes mean when the character type for basic_regex is too small to hold that value? Do we intend to require multi-byte support here (I hope not)? Or is such a value invalid when the target character type is too small?

Resolution:
The general direction is that such a value should be invalid when the target character type is too small. We need specific wording.

7.34 Meaning of the match_partial flag

Submitter: Pete Becker (N1507)
Status: Open

RE.3.1.2 says that the match_partial flag

> Specifies that if no match can be found, then it is acceptable to return a match [from, last) where from!=last, if there exists some sequence of characters [from,to) of which [from,last) is a prefix, and which would result in a full match.

Taking this literally, if I have the expression "a(?=b)(?!b)" and try to match it against "a", the partial match must fail, because the two assertions are contradictory. Is the matcher really required to do this sort of analysis of the expression, and determine that there is no possible continuation that could succeed?

From the name, I would think that partial_match would mean, roughly, that if you reach the end of the search text but are only partway through the regular expression, that's okay. So in the example above, the partial match would succeed. Is that what's intended here?

Comment from John Maddock, on use cases for this feature:

- Searching "infinite" texts: for example two real world use cases that Boost.regex has been put to, are searching a multi-gigabyte server log, and filtering the data passing through a socket. In these cases you can't possibly load all of the text into memory to search it, so you load chunks into a buffer and search one chunk at a time. Then you need to know whether a match could have straddled two chunk boundaries: and that's what a partial match gives you, it tell you how much of the end of one chunk to hang onto before reading the next section.
- Data input validation: if the data in some field has to match some regex to be acceptable, some users what to check this character by character as it's entered - the question then becomes: "given some more input could we eventually match the expression," again that's what a partial match gives you.

This still doesn't give us a specification of the feature, but at least it gives us the motivation.

Resolution:
The proposed resolution in N1507 is to add the non-normative note: "implementations are not required to go to heroic efforts to determine whether a partial match is truly possible." The LWG does not believe this fixes the problem. First, the non-normative note is still vague about what kind of analysis the matcher is supposed to do. Second, a non-normative note can't be a fix for insufficiently precise normative text. It may be that there's no way to specify match_partial precisely enough. If so, this feature should be removed; it may simply be a half-baked feature. Do other regular expression systems have it, and, if so, how do they specify its behavior?

7.35 Name of regex_traits::is_class

Submitter: Pete Becker (N1507)
Status: Open

That name is confusing. I'd prefer inclass, or some variant. The function takes two arguments: a character and a character class, and tells you whether the character belongs to the class. is_class sounds too much like querying whether some object represents a character class.

Resolution: The LWG agrees this isn't a good name. Someone needs to come up with a better one.

7.36 Can traits::error_string be simplified?

Submitter: Pete Becker (N1507)
Status: Open

In the proposal, the template regex_traits has a member function error_string that takes an error code that indicates what error occurred and returns a string corresponding to that error, which is then used as the argument to the constructor for an exception object. Seems to me it would be simpler to have regex_traits simply provide a function that throws the exception, called with the error code. Is this string needed for anything else?

Resolution:

The sense of the LWG is that we should rethink the error reporting policy. A bad_expression object should contain a flag that represents the error, not a string constructed from the flag. The string returned by what() should be left unspecified, and the error_string interface should probably be thrown away entirely. (Programmers who want to test exception objects to find out the exact cause of the error find codes easier to work with than strings. Programmers who want to print diagnostics for users can supply their own code-to-string mechanism.)

7.37 Can traits::translate be improved?

Submitter: Pete Becker (N1507)
Status: Open

The regex_traits member function 'translate' is used when comparing a character in the pattern string with a character in the target string. It takes two arguments: the character to translate, and a boolean flag that indicates whether the translation should be case sensitive. So two characters are equal if

```
translate(pch, icase) == translate(tch, icase)
```

So with pattern text of "abcde", checking for a match would look something like this:

```
for (int i = 0; i < 5; ++i)
   if (translate(pch[i], icase) == translate(tch[i], icase))
      return false;
return true;
```

The implementation of regex_traits::translate in the library-supplied traits class is:

```
return (icase ? use_facet<ctype<charT> >(getloc()).tolower(ch)
: ch);
```

There's potential for a significant speedup, though, if case sensitive and case insensitive comparisons go through two different functions. The obvious transformation of the preceding loop would be:

```
if (icase)
   for (int i = 0; i < 5; ++i)
      if (translate_ic(pch[i]) == translate_ic(tch[i]))
         return false;
else
   for (int i = 0; i < 5; ++i)
      if (translate(pch[i]) == translate(tch[i]))
         return false;
return true;
```

For the default regex_traits class, the calls to translate in the second branch of the if statement would be inline calls to a translate function that simply returns its argument, so the loop turns into a sequence of direct comparisons, with no distractions from the possibility of case insensitivity. Further, since case sensitivity is determined by a flag that's set at the time the regular expression is compiled, one of the two branches of the outer if statement will always be unnecessary.

I made up the names 'translate_ic' and 'translate' for this e-mail. I'm not suggesting that we use them.

Resolution:
We think separating the case-insensitive match from the simple case-sensitive match is probably a good idea. Pete will provide wording for a specific proposal.

7.38 Improving on traits::toi

Submitter: Pete Becker (N1507)
Status: Open

It says, in part:

> If first == last or if is_class(*first, lookup_classname("d")) == false then sets result equal to -1.

And "d" by default is the digits 0-9. Since the radix for the conversion can be 8, 10, or 16, the condition involving "d" isn't right. For a hex value it precludes the value 'a0'. For an octal value it allows '90', but the ensuing conversion will fail. We need to find a different way to express this. The idea is to return -1 on a failed conversion, and the appropriate unsigned value on success.

And further: I'm starting to think that toi is too high level an interface. Regular expression grammars go character bycharcter. For example, the value of a HexEscapeSequence (\xhh) is "(16 times the MV of the first hex digit) plus the MV of the second HexDigit". toi (hypertechnically) doesn't require that. In order to implement the specification literally, the regex parser needs to translate individual characters, not groups of characters, into values, and accumulate those values as appropriate. Thus, regex_traits ought to provide int value(charT ch),which returns -1 if isxdigit(ch) is false, otherwise the numeric value represented by the character.

And: I've just implemented it. Here are the changes I made:

- I removed escape_type_backref and escape_type_decimal
- I added escape_type_numeric (0-9)
- I added int regex_traits::value(charT ch, int base)

The first two aren't technically necessary for this change, but escape_type_backref is a bit misleading. ECMAScriptdoesn't restrict the number of capture groups, so \10 can be a valid back reference. This means thatescape_type_backref alone isn't sufficient. So I figured it's enough to know that you're starting a numeric constant (i.e.escape_type_numeric), and then you can use value() == -1 to determine when you've reached the end of a constant.

The second argument to value is needed in order to decide whether the character is a valid digit for the base. valuereturns -1 for an invalid digit, and the (unsigned) numeric value for a valid digit.

Resolution:
Discussed at Kona. Probably a good idea; the LWG will wait until Pete provides wording.

7.39 Improving on traits::lookup_classname

Submitter: Pete Becker (N1507)
Status: Duplicate

I think this needs a change in specification. It returns a value that identifies the named character class identified by its string argument. The cases I'm concerned about are the ones with names like [:alnum:]. When the code encounters the opening [: it has to scan ahead for the matching :, pick up the characters in between, stuff them into a string, and call lookup_classname. This is a lot of wheel spinning. In particular, creating the string is expensive. If lookup_classname took two iterators instead of a string it could simply look at the characters without the intervening string object.

Resolution:
This is a subset of something the LWG already agreed on in principle: using an iterator interface instead of a string interface. There's no need to discuss this subpart by itself.

7.40 match_results element access functions have incorrect parameter types

Submitter: Robert Klarer
Status: New
Section: 7.9.3 [tr.re.results.acc]

The subscripting operator for match_results is declared as follows:

```
const_reference operator[](int n) const;
```

This declaration is inconsistent with std::vector<...>::operator[], and introduces the possibility that the function may be called incorrectly (using a negative argument).

A similar problem exists for the length(...), position(...), and str(...) members of match_results.

Proposed resolution:

change the declaraction of the subscripting operator for match_results from

```
const_reference operator[](int n) const;
```

to

```
const_reference operator[](size_type n) const;
```

change the declaration of the match_results member function length(...) from

```
difference_type length(int sub = 0) const;
```

to

```
difference_type length(size_type sub = 0) const;
```

change the declaration of the match_results member function position(...) from

```
difference_type position(unsigned int sub = 0) const;
```

to

```
difference_type position(size_type sub = 0) const;
```

change the declaration of the match_results member function str(...) from

```
string_type str(int sub = 0) const;
```

to

```
string_type str(size_type sub = 0) const;
```

7.41 Regex names should be reviewed

Submitter: Matt Austern
Status: New

This is an outgrowth of the Kona discussion of issue 7.13. Names throughout the regex section are rather verbose; this is partly, but not entirely, a result of the regex_ prefix that appears in so many places. We may want to consider a systematic renaming.

8 Fixed-size array issues

8.1 Is "array" the right name?

Submitter: Robert Klarer
Status: New

The name `array` may be confusing, since `array<T>` is not in fact an array; the `is_array` type trait, for example, will return false for `array<T>`. (As it should.) Perhaps another name would make this less surprising.

9 Iterator concept and adapter issues

9.1 iterator_access overspecified?

Submitter: Pete Becker
Status: New

The proposal includes:

enum iterator_access { readable_iterator = 1, writable_iterator = 2, swappable_iterator = 4, lvalue_iterator = 8 };

In general, the standard specifies thing like this as a bitmask type with a list of defined names, and specifies neither the exact type nor the specific values. Is there a reason for iterator_access to be more specific?

9.2 operators of iterator_facade overspecified

Submitter: Pete Becker
Status: New

In general, we've provided operational semantics for things like operator++. That is, we've said that ++iter must work, without requiring either a member function or a non-member function. iterator_facade specifies most operators as member functions. There's no inherent reason for these to be members, so we should remove this requirement. Similarly, some operations are

specified as non-member functions but could be implemented as members. Again, the standard doesn't make either of these choices, and TR1 shouldn't, either. So: operator*(), operator++(), operator++(int), operator--(), operator--(int), operator+=, operator-=, operator-(difference_type), operator-(iterator_facade instance), and operator+ should be specified with operational semantics and not explicitly required to be members or non-members.

9.3 enable_if_interoperable needs standardese

Submitter: Pete Becker
Status: New

The only discussion of what this means is in a note, so is non-normative. Further, the note seems to be incorrect. It says that `enable_if_interoperable` only works for types that "are interoperable, by which we mean they are convertible to each other." This requirement is too strong: it should be that one of the types is convertible to the other.

Proposed resolution:
Remove the enable_if_interoperable stuff, and just write all the comparisons to return bool. Then add a blanket statement that the behavior of these functions is undefined if the two types aren't interoperable.

9.4 enable_if_convertible unspecified, conflicts with requires

Submitter: Pete Becker
Status: New

In every place where `enable_if_convertible` is used it's used like this (simplified):

```
template<class T>
struct C
{
  template<class T1>
  C(T1, enable_if_convertible<T1, T>::type* = 0);
};
```

The idea being that this constructor won't compile if T1 isn't convertible to T. As a result, the constructor won't be considered as a possible overload when constructing from an object x where the type of x isn't convertible to T. In addition, however, each of these constructors has a requires clause that requires convertibility, so the behavior of a program that attempts such a construction is undefined. Seems like the enable_if_convertible part is irrelevant, and should be removed.

There are two problems. First, `enable_if_convertible` is never specified, so we don't know what this is supposed to do. Second: we could reasonably say that this overload should be disabled in certain cases or we could reasonably say that behavior is undefined, but we can't say both.

Thomas Witt writes that the goal of putting in `enable_if_convertible` here is to make sure that a specific overload doesn't interfere with the generic case except when that overload makes sense. He agrees that what we currently have is deficient.

Dave Abrahams writes that there is no conflict with the requires cause "because the requires

clause only takes effect when the function is actually called. The presence of the constructor signature
can/will be detected by is_convertible without violating the requires clause, and thus it makes a difference to disable those constructor instantiations that would be disabled by enable_if_convertible even if calling them invokes undefined behavior."

There was more discussion on the reflector: c++std-lib-12312, c++std-lib-12325, c++std-lib-12330, c++std-lib-12334, c++std-lib-12335, c++std-lib-12336, c++std-lib-12338, c++std-lib-12362.

Proposed resolution:
Specify enable_if_convertible to be as-if:

```
template <bool> enable_if_convertible_impl
{};

template <> enable_if_convertible_impl<true>
{ struct type; };

template<typename From, typename To>
struct enable_if_convertible
  : enable_if_convertible_impl<
        is_convertible<From, To>::value
{};
```

9.5 iterator_adaptor has an extraneous 'bool' at the start of the template definition

Submitter: Pete Becker
Status: New

The title says it all; this is probably just a typo.

9.6 Name of private member shouldn't be normative

Submitter: Pete Becker
Status: New

iterator_adaptor has a private member named m_iterator. Presumably this is for exposition only, since it's an implementation detail. It needs to be marked as such.

9.7 iterator_adaptor operations specifications are a bit inconsistent

Submitter: Pete Becker
Status: New

iterator_adpator() has a Requires clause, that Base must be default constructible.
iterator_adaptor(Base) has no Requires clause, although the Returns clause says that the Base member is copy construced from the argument (this may actually be an oversight in N1550, which doesn't require iterators to be copy constructible or assignable).

9.8 Specialized adaptors text should be normative

Submitter: Pete Becker
Status: New

similar to 9.3, "Specialized Adaptors" has a note describing enable_if_convertible. This should be normative text.

9.9 Reverse_iterator text is too informal

Submitter: Pete Becker
Status: New

reverse iterator "flips the direction of the base iterator's motion". This needs to be more formal, as in the current standard. Something like: "iterates through the controlled sequence in the opposite direction"

9.10 “prior” is undefined

Submitter: Pete Becker
Status: New

reverse_iterator::dereference is specified as calling a function named 'prior' which has no specification.

9.11 “In other words” is bad wording

Submitter: Pete Becker
Status: New

Transform iterator has a two-part specification: it does this, in other words, it does that. "In other words" always means "I didn't say it right, so I'll try again." We need to say it once.

9.12 Transform_iterator shouldn’t mandate private member

Submitter: Pete Becker
Status: New

transform_iterator has a private member named 'm_f' which should be marked "exposition only."

9.13 Unclear description of counting iterator

Submitter: Pete Becker
Status: New

The description of Counting iterator is unclear. "The counting iterator adaptor implements dereference by returning a reference to the base object. The other operations are implemented by the base m_iterator, as per the inheritance from iterator_adaptor."

9.14 Counting_iterator’s difference type

Submitter: Pete Becker
Status: New

Counting iterator has the following note:

> [Note: implementers are encouraged to provide an implementation of distance_to and a difference_type that avoids overflows in the cases when the Incrementable type is a numeric type.]

I'm not sure what this means. The user provides a template argument named Difference, but there's no difference_type. I assume this is just a glitch in the wording. But if implementors are encouraged to ignore this argument if it won't work right, why is it there?

9.15 How to detect lvalueness?

Submitter: Dave Abrahams
Status: New

Shortly after N1550 was accepted, we discovered that an iterator's lvalueness can be determined knowing only itsvalue_type. This predicate can be calculated even for old-style iterators (on whose reference type the standard places few requirements). A trait in the Boost iterator library does it by relying on the compiler's unwillingness to bind an rvalue to a T& function template parameter. Similarly, it is possible to detect an iterator's readability knowing only itsvalue_type. Thus, any interface which asks the user to explicitly describe an iterator's lvalue-ness or readability seems to introduce needless complexity.

9.16 is_writable_iterator returns false positives

Submitter: Dave Abrahams
Status: New

is_writable_iterator returns false positives for forward iterators whose value_type has a private assignment operator, or whose reference type is not a reference (currently legal).

9.17 is_swappable_iterator returns false positives

Submitter: Dave Abrahams
Status: New

is_swappable_iterator has the same problems as is_writable_iterator. In addition, if we allow users to write their own iter_swap functions it's easy to imagine old-style iterators for which is_swappable returns false negatives.

9.18 Are is_readable, is_writable, and is_swappable useful?

Submitter: Dave Abrahams
Status: New

I am concerned that there is little use for any of is_readable, is_writable, or is_swappable, and that not only do they unduly constrain iterator implementors but they add overhead to iterator_facade and iterator_adaptor in the form of a template parameter which would otherwise be unneeded. Since we can't implement two of them accurately for old-style iterators, I am having a hard time justifying their impact on the rest of the proposal(s).

9.19 Non-Uniformity of the "lvalue_iterator Bit"

Submitter: Dave Abrahams
Status: New

The proposed iterator_tag class template accepts an "access bits" parameter which includes a bit to indicate the iterator's lvalueness (whether its dereference operator returns a reference to its value_type. The relevant part of N1550 says:

> The purpose of the lvalue_iterator part of the iterator_access enum is to communicate to iterator_tagwhether the reference type is an lvalue so that the appropriate old category can be chosen for the base class. The lvalue_iterator bit is not recorded in the iterator_tag::access data member.

The lvalue_iterator bit is not recorded because N1550 aims to improve orthogonality of the iterator concepts, and a new-style iterator's lvalueness is detectable by examining its reference type. This inside/outside difference is awkward and confusing.

9.20 Traversal Concepts and Tags

Submitter: Dave Abrahams
Status: New

Howard Hinnant pointed out some inconsistencies with the naming of these tag types:

```
incrementable_iterator_tag            // ++r, r++
single_pass_iterator_tag              // adds a == b, a != b
forward_traversal_iterator_tag        // adds multi-pass
bidirectional_traversal_iterator_tag  // adds --r, r--
random_access_traversal_iterator_tag  // adds r+n,n+r,etc.
```

Howard thought that it might be better if all tag names contained the word "traversal".

It's not clear that would result in the best possible names, though. For example, incrementable iterators can only make a single pass over their input. What really distinguishes single pass iterators from incrementable iterators is not that they can make a single pass, but that they are equality comparable. Forward traversal iterators really distinguish themselves by introducing multi-pass capability. Without entering a "Parkinson's Bicycle Shed" type of discussion, it might be worth giving the names of these tags (and the associated concepts) some extra attention.

9.21 iterator_facade Derived template argument underspecified

Submitter: Pete Becker
Status: New

The first template argument to iterator_facade is named Derived, and the proposal says:

> The Derived template parameter must be a class derived from iterator_facade.

First, iterator_facade is a template, so cannot be derived from. Rather, the class must be derived from a specialization of iterator_facade. More important, isn't Derived required to be the class

that is being defined? That is, if I understand it right, the definition of D here this is not valid:

class C : public iterator_facade<C, ... > { ... };

class D : public iterator_facade<C, ...> { ... };

In the definition of D, the Derived argument to iterator_facade is a class derived from a specialization of iterator_facade, so the requirement is met. Shouldn't the requirement be more like "when using iterator_facade to define an iterator class Iter, the class Iter must be derived from a specialization of iterator_facade whose first template argument is Iter." That's a bit awkward, but at the moment I don't see a better way of phrasing it.

9.22 return type of Iterator difference for iterator facade

Submitter: Pete Becker
Status: New

The proposal says:

```
template <class Dr1, class V1, class AC1, class TC1, class R1, class D1,
          class Dr2, class V2, class AC2, class TC2, class R2, class D2>
typename enable_if_interoperable<Dr1, Dr2, bool>::type
operator -(iterator_facade<Dr1, V1, AC1, TC1, R1, D1> const& lhs,
           iterator_facade<Dr2, V2, AC2, TC2, R2, D2> const& rhs);
```

Shouldn't the return type be one of the two iterator types? Which one? The idea is that if one of the iterator types can be converted to the other type, then the subtraction is okay. Seems like the return type should then be the type that was converted to. Is that right?

9.23 Iterator_facade: minor wording Issue

Submitter: Pete Becker
Status: New

In the table that lists the required (sort of) member functions of iterator types that are based on iterator_facade, the entry for c.equal(y) says:

> true iff c and y refer to the same position. Implements c == y and c != y.

The second sentence is inside out. c.equal(y) does not implement either of these operations. It is used to implement them. Same thing in the description of c.distance_to(z).

9.24 Use of undefined name in iterator_facade table

Submitter: Pete Becker
Status: New

Several of the descriptions use the name X without defining it. This seems to be a carryover from the table immediately above this section, but the text preceding that table says "In the table below, X is the derived iterator type." Looks like the X:: qualifiers aren't really needed;

X::reference can simply be reference, since that's defined by the iterator_facade specialization itself.

9.25 Iterator_facade: wrong return type

Submitter: Pete Becker
Status: New

Several of the member functions return a Derived object or a Derived&. Their Effects clauses end with:

```
return *this;
```

This should be

```
return *static_cast<Derived*>(this);
```

9.26 Iterator_facade: unclear returns clause for operator[]

Submitter: Pete Becker
Status: New

The returns clause for operator[](difference_type n) const says:

> Returns: an object convertible to X::reference and holding a copy p of a+n such that, for a constant object v of type X::value_type, X::reference(a[n] = v) is equivalent to p = v.

This needs to define 'a', but assuming it's supposed to be *this (or maybe *(Derived*)this), it still isn't clear what this says. Presumably, the idea is that you can index off of an iterator and assign to the result. But why the requirement that it hold a copy of a+n? Granted, that's probably how it's implemented, but it seems over-constrained. And the last phrase seems wrong. p is an iterator; there's no requirement that you can assign a value_type object to it. Should that be *p = v? But why the cast in reference(a[n] = v)?

9.27 Iterator_facade: redundant clause

Submitter: Pete Becker
Status: New

operator- has both an effects clause and a returns clause. Looks like the returns clause should be removed.

9.28 indirect_iterator: incorrect specification of default constructor

Submitter: Pete Becker
Status: New

The default constructor returns "An instance of indirect_iterator with a default constructed base object", but the constructor that takes an Iterator object returns "An instance of indirect_iterator with the iterator_adaptor subobject copy constructed from x." The latter is the correct form, since it does not reach inside the base class for its semantics. So the default constructor shoudl return

"An instance of indirect_iterator with a default-constructed iterator_adaptor subobject."

9.29 indirect_iterator: unclear specification of template constructor

Submitter: Pete Becker
Status: New

The templated constructor that takes an indirect_iterator with a different set of template arguments says that it returns "An instance of indirect_iterator that is a copy of [the argument]". But the type of the argument is different from the type of the object being constructed, and there is no description of what a "copy" means. The Iterator template parameter for the argument must be convertible to the Iterator template parameter for the type being constructed, which suggests that the argument's contained Iterator object should be converted to the target type's Iterator type. Is that what's meant here?

(Pete later writes: In fact, this problem is present in all of the specialized adaptors that have a constructor like this: the constructor returns "a copy" of the argument without saying what a copy is.)

9.30 transform_iterator argument irregularity

Submitter: Pete Becker
Status: New

The specialized adaptors that take both a Value and a Reference template argument all take them in that order, i.e. Value precedes Reference in the template argument list, with the exception of transform_iterator, where Reference precedes Value. This seems like a possible source of confusion. Is there a reason why this order is preferable?

9.31 function_output_iterator overconstrained

Submitter: Pete Becker
Status: New

function_output_iterator requirements says: "The UnaryFunction must be Assignable, Copy Constructible, and the expression f(x) must be valid, where f is an object of type UnaryFunction and x is an object of a type accepted by f."

Everything starting with "and," somewhat reworded, is actually a constraint on output_proxy::operator=. All that's needed to create a function_output_iterator object is that the UnaryFunction type be Assignable and CopyConstructible. That's also sufficient to dereference and to increment such an object. It's only when you try to assign through a dereferenced iterator that f(x) has to work, and then only for the particular function object that the iterator holds and for the particular value that is being assigned.

9.32 Should output_proxy really be a named type?

Submitter: Pete Becker
Status: New

This means someone can store an output_proxy object for later use, whatever that means. It also

constrains output_proxy to hold a copy of the function object, rather than a pointer to the iterator object. Is all this mechanism really necessary?

9.33 istreambuf_iterator isn't a Readable Iterator

Submitter: Pete Becker
Status: New

c++std-lib-12333:
N1550 requires that for a Readable Iterator a of type X, *a returns an object of type iterator_traits<X>::reference. istreambuf_iterator::operator* returns charT, but istreambuf_iterator::reference is charT&. So am I overlooking something, or is istreambuf_iterator not Readable?