Make Random Number Engines Seedable

Document #:	D2060R0
Date:	2020-01-13
Project:	Programming Language C++
Audience:	LEWG

Reply-to: Martin Hořeňovský <martin.horenovsky@gmail.com>

Introduction

The C++ standard library provides a set of utilities, that is random number engines, statistical distributions and seeding utilities, to generate pseudo-random numbers with different properties. Unfortunately, the current state of seeding random number engines is fundamentally broken.

Motivation

A random number engine as defined in the C++ standard can be seeded in one of 2 ways:

- via a single value of engine's `result_type`
- via a type that satisfies the *SeedSequence* named requirement, such as `std::seed_seq`

The first option is commonly used together with `std::random_device` to get a random number engine with random initial state, like so:

```
#include <iostream>
#include <random>

int main() {
    std::mt19937 urbg(std::random_device{}());
    std::cout << urbg() << '\n';
}
```

This code works and *should*[1] print out a different number every run. However, there is a massive difference between the seed size, which is specified to be `unsigned int` and thus 32 bits on most commonly used platforms, and the size of a Mersenne Twister's internal state, which is 624 32-bit unsigned integers.

A side effect of this kind of insufficiently large random seeding is that if I tell you that the first number

[1] Assuming good implementation of `random_device`

generated by `urbg` is 3046098682[2], you can quickly[3] find the seed and use it to predict all future outputs.

In theory, this can be fixed by using a *SeedSequence*, such as the standard-provided `seed_seq`, but there are two problems with this:

1. We have no way to know how much randomness we have to feed into a `seed_seq` to initialize a specific engine, and
2. `seed_seq` is not a bijection even if you give it enough randomness for target data storage

The second problem can be demonstrated with a simple example[4]:

```
#include <array>
#include <iostream>
#include <random>

int main() {
    std::seed_seq seq1({0xf5e5b5c0, 0xdcb8e4b1}),
                  seq2({0xd34295df, 0xba15c4d0});

    std::array<uint32_t, 2> arr1, arr2;
    seq1.generate(arr1.begin(), arr1.end());
    seq2.generate(arr2.begin(), arr2.end());

    std::cout << (arr1 == arr2) << '\n';
}
```

This code outputs `1`, which shows that two different instances of `std::seed_seq`, given different 64 bits of seeding, generate the same 64 bits of output.

These issues mean that properly seeding the Random Number Engines in `<random>` is impossible, which should be fixed.

Proposed solution

User should be able to retrieve generator's seed size

To solve the problem with the full size of a seed for a a random number engine being unknowable, I propose adding a new requirement on a random number engine in [rand.req.eng]. Specifically, a member function `static constexpr size_t required_seed_size()` that returns the number of bytes required to fully seed random number engine of that type.

[2] Another interesting thing is that there are no 32 bit seeds that will give you 7 as the first output from `urbg`. Also, 7 is not the only number with this property.

[3] It took about 10 minutes on my desktop PC

[4] The values are borrowed from a blogpost by Melissa E. O'Neill of PCG

Introduce new standard types that conform to the *SeedSequence* requirements

To solve the problems with `std::seed_seq`, I propose adding new type that conforms to the *SeedSequence* requirement to the standard library.

Specifically, I propose adding a type to the standard, `sized_seed_seq`, whose `generate` member function writes out the data exactly as they were passed to the instance during construction. If the amount of data requested by user from `generate` is less than the amount of data stored, only the first n bytes is written out. If the amount of data requested is larger, the behaviour is undefined.

By combining the two changes above, we can now properly seed a random number engine, such as `mt19337`, with random data, using code like this:

```
#include <algorithm>
#include <array>
#include <iostream>
#include <random>

int main() {
    std::array<uint32_t, std::mt19337::required_seed_size()> random_data;
    std::generate(random_data.begin(), random_data.end(), std::random_device{});
    std::mt19937 urbg(std::sized_seed_seq(random_data.begin(), random_data.end()));
    std::cout << urbg() << '\n';
}
```

However, while proper random seeding is now possible, the code is quite long and clunky. Because randomly initializing a random number engine is a common use case, I propose adding `generate` member function to `random_device`, so it can be used as a *SeedSequence*. This would transform the code above into this:

```
#include <iostream>
#include <random>

int main() {
    std::mt19937 urbg(std::random_device{});
    std::cout << urbg() << '\n';
}
```

With this change, it is easier to seed a random number engine with random seed correctly, than it is to seed it incorrectly. However, `random_device` does not, and can not, fulfill the *SeedSequence* requirements. Because of this I also suggest refining the requirements in the next part.

Refine *SeedSequence* requirement

To allow for the type `std::random_seed_seq` to integrate with the rest of the random facilities properly, I propose renaming the *SeedSequence* named requirement to *RepeatableSeedSeq*, and introducing a weaker requirement using the old name of *SeedSequence*. This weaker requirement no longer includes the `size`

and the `param` member functions, and also removes the repeatability requirements on `generate` called with the same arguments.

Alternative approach

An alternative approach to changing `std::random_device` to act as a *SeedSequence* is to introduce new type, named `random_seed_seq`, and have it serve as a *SeedSequence* whose `generate` writes out high-quality random bits.

Goals

To recapitulate, this paper has three goals:

1. Make it possible for generic code to determine how big a seed does a random number engine require.
2. Add a new type satisfying the requirements of *SeedSequence* to the standard library, one that avoids some of the problems with `std::seed_seq`.
3. Improve the usability of seeding by adding a *SeedSequence* type that seeds with high-quality random bits, sacrificing repeatability.

Acknowledgments

I want to thank Melissa E. O'Neill whose work I drew upon for this paper.