Discouraging **rand()** in C++14

Document #: WG21 N3841
Date: 2014-01-01
Revises: N3775
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

In their final Chicago deliberations re [N3775] vis-à-vis National Body comment US21, LEWG and LWG achieved joint consensus (1) to deprecate **std::random_shuffle** for C++14 as proposed, and (2) to strengthen the existing Note in [c.math]/5 in order to further encourage **rand()** users to migrate to the **<random>** component of the C++11 standard library. This paper provides wording to implement these decisions.

1 Background and proposal

> *If a feature is not deprecated [I] don't see any point in not using it.*
> — Hariharan Subramanian

By common consensus at several consecutive WG21 meetings during which the C++11 random number facility was being discussed and shaped into its final form, it has for a number of years been the long-term plan to excise the legacy C random number facility (made up of functions **rand** and **srand** and of macro **RAND_MAX**). Indeed, WG21 voted several years ago to insert a Note[1] into [c.math]/5 as a head start on this plan: "The random number generation (26.5) facilities in this standard are often preferable to **rand**."[2]

Throughout deliberations in Chicago vis-à-vis National Body comment US21, LEWG and LWG independently agreed that we should continue to encourage **rand()** users to migrate to the **<random>** component of the C++11 standard library.[3] Taking into account feedback received from WG21, LEWG and LWG achieved a joint final consensus to address US21 by making two adjustments to the text of the C++14 draft standard:

[1] This language originated with Beman Dawes in [N2669]; [N2691] was the first Working Paper to incorporate it.

[2] See also Stephan T. Lavavej's talk, "**rand()** Considered Harmful," given at the GoingNative 2013 event. Recorded on 2013-09-06; available at http://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful.

[3] Readers seeking greater familiarity with this component may find [N3551] to be a helpful source of background information and tutorial guidance with numerous usage examples.

1. Strengthen the existing Note, quoted above, in [c.math]/5.
2. Deprecate **std::random_shuffle** as proposed in [N3775] because "one overload is specified so as to depend on **rand**, while the other overload is specified so as to require a hard-to-produce distribution object from the user; such a distribution is already an implicit part of **shuffle**, which we retain."

The next section proposes wording to implement both parts of this decision.

2 Proposed wording[4]

(1) Augment [c.math]/5 as shown. (The added wording is adapted from the introductory section of [N3551].)

5 . . . . [*Note:* The random number generation (26.5) facilities in this standard are often preferable to **rand**, as **rand**'s underlying algorithm is unspecified. Use of **rand** therefore continues to be nonportable, with unpredictable and oft-questionable quality and performance. *— end note*]

(2) Copy all of the current [alg.random.shuffle] to a new section in Annex D, applying to the copy the changes shown below.

D.x Random shuffle [depr.alg.random.shuffle]

The function templates **random_shuffle** are deprecated.

```
template<class RandomAccessIterator>
  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);

template<class RandomAccessIterator, class RandomNumberGenerator>
  void random_shuffle(RandomAccessIterator first, RandomAccessIterator last,
                      RandomNumberGenerator&& ~~rand~~rng);

~~template<class RandomAccessIterator, class UniformRandomNumberGenerator>~~
  ~~void shuffle(RandomAccessIterator first, RandomAccessIterator last,~~
               ~~UniformRandomNumberGenerator&& g);~~
```

Effects: Permutes the elements in the range [**first**, **last**) such that each possible permutation of those elements has equal probability of appearance.

Requires: **RandomAccessIterator** shall satisfy the requirements of **ValueSwappable** (17.6.3.2). The random number generating function object ~~**rand**~~**rng** shall have a return type that is convertible to **iterator_traits<RandomAccessIterator>::difference_type**, and the call ~~**rand**~~**rng(n)** shall return a randomly chosen value in the interval [0, **n**), for **n** > 0 of type **iterator_traits<RandomAccessIterator>::difference_type**. ~~The type UniformRandomNumberGenerator shall meet the requirements of a uniform random number generator (26.5.1.3) type whose return type is convertible to iterator_traits<RandomAccessIterator>::difference_type.~~

Complexity: Exactly (**last** − **first**) − 1 swaps.

Remarks: To the extent that the implementation of these functions makes use of random numbers, the implementation shall use the following sources of randomness:

[4] All proposed additions and ~~deletions~~ are relative to the post-Chicago Working Draft [N3797]. Editorial notes are displayed against a gray background. We make no recommendation for any SG10 feature-test macro, as no feature is being added or removed.

The underlying source of random numbers for the first form of the function is implementation-defined. An implementation may use the **`rand`** function from the standard C library.

In the second form of the function, the function object ~~**`rand`**~~**`rng`** shall serve as the implementation's source of randomness.

~~In the third `shuffle` form of the function, the object `g` shall serve as the implementation's source of randomness.~~

(3) In the synopsis in [algorithms.general]:

- apply the comment ***`//Deprecated`*** to each of the two declarations of **`random_shuffle`**;
- at the Project Editor's discretion, append to these same declarations a cross-reference to the new Annex D section [depr.alg.random.shuffle];
- change the parameter name **`rand`** to **`rng`** in the second of the two declarations of **`random_shuffle`** so as to avoid confusion with the C library function **`rand`**; and
- change the parameter name **`rand`** to **`g`** in the declaration of **`shuffle`** so as to make this declaration consistent with that in **`shuffle`**'s later exposition.

(4) Finally, excise vestiges of **`std::random_shuffle`** from [alg.random.shuffle] by adjusting as follows:

25.3.12 ~~Random s~~Shuffle [alg.~~random.~~shuffle]

~~`template<class RandomAccessIterator>`~~
~~`void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);`~~

~~`template<class RandomAccessIterator, class RandomNumberGenerator>`~~
~~`void random_shuffle(RandomAccessIterator first, RandomAccessIterator last,`~~
~~`RandomNumberGenerator&& rand);`~~

```
template<class RandomAccessIterator, class UniformRandomNumberGenerator>
  void shuffle(RandomAccessIterator first, RandomAccessIterator last,
               UniformRandomNumberGenerator&& g);
```

Effects: Permutes the elements in the range [**`first`**, **`last`**) such that each possible permutation of those elements has equal probability of appearance.

Requires: **`RandomAccessIterator`** shall satisfy the requirements of **`ValueSwappable`** (17.6.3.2). ~~The random number generating function object `rand` shall have a return type that is convertible to `iterator_traits<RandomAccessIterator>::difference_type`, and the call `rand(n)` shall return a randomly chosen value in the interval [0, `n`), for n > 0 of type `iterator_traits<RandomAccessIterator>::difference_type`.~~ The type **`UniformRandomNumberGenerator`** shall meet the requirements of a uniform random number generator (26.5.1.3) type whose return type is convertible to **`iterator_traits<RandomAccessIterator>::difference_type`**.

Complexity: Exactly (**`last`** − **`first`**) − 1 swaps.

Remarks: To the extent that the implementation of ~~these~~ this function~~s~~ makes use of random numbers, ~~the implementation shall use the following sources of randomness:~~

~~The underlying source of random numbers for the first form of the function is implementation-defined. An implementation may use the `rand` function from the standard C library.~~

~~In the second form of the function, the function object `rand` shall serve as the implementation's source of randomness.~~

~~In the third shuffle form of the function,~~ the object **g** shall serve as the implementation's source of randomness.

3 Acknowledgments

Many thanks, for their thoughtful comments, to Stephan T. Lavavej and the other reviewers of early drafts of this paper.

4 Bibliography

[N2669] Beman Dawes et al.: "Thread-Safety in the Standard Library (Rev 2)." ISO/IEC JTC1/SC22/WG21 document N2669 (post-Sophia mailing), 2008-06-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2669.pdf.

[N2691] Pete Becker: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N2691 (post-Sophia mailing), 2008-06-27. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2008/n2691.pdf.

[N3551] Walter E. Brown: "Random Number Generation in C++11." ISO/IEC JTC1/SC22/WG21 document N3551 (pre-Bristol mailing), 2013-03-12. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf.

[N3742] Walter E. Brown: "Three **<random>**-related Proposals, v2." ISO/IEC JTC1/SC22/WG21 document N3742 (pre-Chicago mailing), 2013-08-30. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3742.pdf.

[N3775] Walter E. Brown: "Deprecating **rand()** and Friends." ISO/IEC JTC1/SC22/WG21 document N3775 (post-Chicago mailing), 2013-09-25. Revises part of [N3742]. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3775.pdf.

[N3797] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3797 (post-Chicago mailing), 2013-10-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf.

5 Revision history

Version	**Date**	**Changes**
1	2014-01-01	• Published as N3841.