Subtract With Carry Engine Correction Proposal

Visual inspection of various features of the framework

Document #:	D0000R1
Date:	2025-04-28
Project:	Programming Language C++
Audience:	Library Evolution Library
Revises:	D0000R0

Reply-to:	Author Juan Lucas Rey <juanlucasrey@gmail.com>

1 Introduction

This proposal addresses a flaw in the current behavior of `subtract_with_carry_engine`, whereby distinct internal states can produce identical sequences of outputs. This state degeneracy leads to surprising and non-intuitive behavior, violating expected properties such as equality of random engines based on observable output.

2 Motivation and Problem Description

The `subtract_with_carry_engine` maintains an internal state sequence along with a carry bit. Due to the mathematical structure of the engine, it is possible for two different internal states to produce identical sequences of random numbers.

This phenomenon indicates an over-parametrization of the state: distinct internal representations that are functionally equivalent with respect to output.

Consequently, two engine instances `rng1` and `rng2` may produce identical random sequences while `rng1 != rng2` according to the `==` operator.

An example of this problematic behavior is shown in this test case.

Such behavior is problematic for users expecting engine equality to reflect output behavior and can lead to incorrect assumptions or errors when engines are used in associative containers or algorithm comparisons.

3 Proposed Resolution

We propose clarifying that engines that generate identical sequences should be considered equal, and where feasible, that the internal state should normalize itself after a full cycle or by lazy canonicalization during comparisons.

Specifically:

— Define a “canonical form” of the internal state after generating a full cycle.
— Specify that two engines are `==` if their observable sequences are identical.
— Optionally recommend that implementations normalize state representations after output cycles to facilitate efficient equality comparison.

4 Impact on Existing Implementations

This change primarily impacts engine comparison operations and serialization/deserialization use cases.

Engines already relying purely on observable sequences would not be affected. Implementations that rely on byte-for-byte internal state equality would require adjustment.

No impact is expected on random number output sequences themselves.

5 References

— Reference test case highlighting the issue