LWG Issue 2168 is NAD

Document #: WG21 N3926
Date: 2014-02-14
Revises: None
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

LWG issue 2168 reports a perceived inconsistency between two parts of the library's specification for **`uniform_real_distribution`**. This paper argues that there is no inconsistency and that the issue should therefore be closed as NAD.

1 Introduction

Library Working Group issue 2168 says:

> uniform_real says in 26.5.8.2.2 [rand.dist.uni.real] p1:
>
> > A **`uniform_real_distribution`** random number distribution produces random numbers x, a $\leq$ x $<$ b,
>
> but also that (26.5.8.2.2 [rand.dist.uni.real] p2):
>
> > **`explicit uniform_real_distribution(RealType a=0.0, RealType b=1.0);`**
> > -2- Requires: a $\leq$ b and b - a $\leq$ **`numeric_limits<RealType>::max()`**.
>
> If you construct a **`uniform_real_distribution<RealType>(a, b)`** where there are no representable numbers between 'a' and 'b' (using **`RealType`**'s representation) then you cannot satisfy 26.5.8.2.2 [rand.dist.uni.real]. An obvious example is when **`a == b`**.

Despite the perceived inconsistency reported by the issue, we will argue in the next section that there is in fact no defect, and that the issue should therefore be closed as NAD.

2 Analysis and recommendation

The most recent change to the behaviorial specification of **`uniform_real_distribution`** seems to have been in 2006. Before that change, the requirement on the **`operator()`** functions was that they produce values x such that $a < x < b$. Since then, we have instead required $a \leq x < b$, as the issue correctly cites. We had made that change in order to cater to requests from programmers who, at least in our C++ world, tend to be more comfortable with half-open intervals than they are with the fully open intervals that statisticians and mathematicians often prefer.[1]

[1] And we are after all, programmers!

Please note that under no circumstances would we ever, ever, ever consider having a closed interval here. Sadly, the issue quotes the C++ standard out of context on this point, failing to cite the adjoining mathematics that carefully specifies the requirement on the associated probability density function, amely that it have the constant value given by $1/(b-a)$. This means that users must ensure that $a \neq b$ whenever **`operator()`** is called, else the resulting behavior is as undefined as is mathematical division by zero.[2] This is not an accident, and must not be changed.[3]

However, we don't need the same strict $a \neq b$ requirement when constructing an object of a **`uniform_real_distribution`** type. Such an object can live perfectly well with equal values of a and b. (Actually, we could have even permitted $b < a$, but that would have incurred unnecessary complexity and performance cost for implementations when calculating the behind-the-scenes scale factor. Based on this engineering assessment, we carefully crafted the user requirement to allow implementations to use the simple and cheap subtraction $b - a$.)

It can, of course, be argued that an object having $a = b$ is inherently impotent, since it can never satisfy the precondition of its **`operator()`** and hence ought never be called. However, we had this argument many years ago, and it turns out to be false: It may be less well known, but there is in fact a well-specified overload of **`operator()`** that ignores the values with which the invoking object was constructed.

Consider the following code fragment, which ought to compile happily after supplying appropriate header, namespace, and other context:

```
1  using dist_t = uniform_real_distribution<>;
2  using parm_t = typename dist_t::param_type;
3
4  default_random_engine e{};
5  dist_t d{0.0, 0.0};  // so far so good
6
7  auto variate = d(e, parm_t{4.5, 6.78});
8    // in this call, ignore the distribution param values
9    // that were supplied via d's c'tor, and instead use
10   // the param values supplied here via parm_t's c'tor
```

There seems no reason to forbid such code, as it is perfectly well-formed, well-specified, well-behaved, and incredibly useful in applications whose distribution's parameters are in flux over time. [4] Moreover, the respective specified preconditions for constructing and for invoking such a distribution object are consistent with each other.

We therefore recommend that the cited issue report be closed as Not A Defect, with the discussion simply pointing to this paper by way of rationale.

3 Revision history

Version	Date	Changes
1	2014-02-14	• Published as N3926.

[2] "Black holes are where God divided by zero." — Steven Wright ☺

[3] Implementors should therefore simply follow their existing/prevailing policy re users who violate a documented precondition: **`terminate()`**, or **`assert()`** first, or **`throw`**, or enter an infinite loop, or return a predetermined out-of-bandwidth value, or do whatever. But such actions are at best a courtesy to users; as we all know, the C++ standard does not specify what happens when a precondition is violated.

[4] Algorithm **`std::shuffle`** exhibits such behavior, for example, although it employs **`uniform_int_distribution`** rather than **`uniform_real_distribution`**.