An Incremental Improvement to `integral_constant`

Document #: WG21 N3545
Date: 2013-03-12
Revises: None
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper proposes to add a `constexpr operator()` to the synopsis of `integral_constant` in order to allow the template to serve as a source of compile-time function objects.

1 Background

[Mad03], a revision of its predecessor [Mad02], was the earliest WG21 proposal to mention `integral_constant`, a class template serving principally as a type wrapper for a compile time constant value. WG21 ultimately accepted that template, with associated typedefs exactly as proposed, for publication in TR1 [ISO07] and thence incorporated it into C++0X working draft [Beck06].

Since then, the only changes to the template were introduced by [Mer09a], which broadly applied `constexpr` throughout the standard library. In the case of `integral_constant`, that paper not only emended the template's `static const` data member to a `static constexpr` one, but also injected a conversion operator:[1]

```
constexpr operator value_type() { return value; }
```

Approved by WG21 via the successor paper [Mer09b], these updates made their first C++0X appearance in working draft [Beck09].

2 Proposal

We propose a further increase in the template's applicability and utility by injecting an `operator()` member that returns the value of the template's data member. The presence of this member function will allow creation of function objects of `integral_constant<>` type. Moreover, we propose to allow such use at compile time by declaring this new member as a `constexpr` function. As a result, each of the many type traits that inherit from `integral_constant` (or from its associated typedefs `true_type` and `false_type`) will also be usable in like manner implicitly:

[1] Alas, the paper provided neither rationale nor discussion for this addition.

```
1 ··· std::is_arithmetic<T>::value ···                  // per TR1 & C++11
2 ··· static_cast<bool>(std::is_arithmetic<T>{}) ···    // per C++11
3 ··· std::is_arithmetic<T>{}() ···                     // as proposed
```

3 Discussion

We view this proposal as completing the **integral_constant** adjustments begun in [Mer09a, Mer09b]. Those amendments, as intended, made it possible for **integral_constant**'s users to take advantage of C++11's new **constexpr** features. However, only contexts that perform implicit conversion (to **bool**, typically, as provided in [conv]/4) are able to make full and effective use of the added capability. Outside such contexts, an explicit **static_cast** or equivalent is needed.

The standard library's **enable_if** and **conditional** type traits exemplify contexts that do not engender such implicit conversion. At the same time, these are commonly-used, rich sources of oft-sophisticated type traits applications that could benefit from an even slightly improved syntax. We prefer to encourage straightforward coding over such circumlocutions as casts and double negations to achieve an equivalent purpose. Accordingly, we recommend the present proposal for earnest consideration by WG21.

4 Proposed wording

Above [meta.help]/1 in WG21 draft [DuT12], add the green text to the synopsis of **integral_constant** as shown below. (Extra blank lines have been inserted for improved legibility; their adoption is at the discretion of the Project Editor. We have also rephrased the nested **typedef**s using C++11 type alias syntax; adoption of this style is also at the Editor's discretion.)

```
template <class T, T v>
struct integral_constant {
  static constexpr T value = v;

  using value_type = T;
  using type = integral_constant<T,v>;

  constexpr operator value_type() { return value; }
  constexpr value_type operator()() { return value; }
};
```

5 Acknowledgments

Many thanks, for their insightful comments, to the readers of early drafts of this paper.

6 Bibliography

[Beck06] Pete Becker: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N2009 (post-Berliln mailing), 2006-04-21. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n2009.pdf.

[Beck09] Pete Becker: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3000 (post-Santa-Cruz mailing), 2009-11-09. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3000.pdf.

[DuT12] Stefanus Du Toit: "Working Draft, Standard for Programming Language C++." ISO/IEC JTC1/SC22/WG21 document N3485 (post-Portland mailing), 2012-11-02.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3485.pdf.

[ISO07] International Organization for Standardization: "Information technology — Programming languages — Technical Report on C++ Library Extensions." ISO/IEC document TR 19768:2007.

[Mad02] John Maddock: "A Proposal to add Type Traits to the Standard Library." ISO/IEC JTC1/SC22/WG21 document N1345 (pre-Curacao mailing), 2002-03-07.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2002/n1345.htm.

[Mad03] John Maddock: "A Proposal to add Type Traits to the Standard Library." ISO/IEC JTC1/SC22/WG21 document N1424 (pre-Oxford mailing) revising [Mad02], 2003-03-03.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2003/n1424.htm.

[Mer09a] Alisdair Meredith: "**`constexpr`** in the libray [*sic*]: take 2." ISO/IEC JTC1/SC22/WG21 document N2976 (pre-Santa-Cruz mailing), 2009-09-22.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2976.html.

[Mer09b] Alisdair Meredith: "**`constexpr`** in the libray [*sic*]: take 2." ISO/IEC JTC1/SC22/WG21 document N2994 (post-Santa-Cruz mailing) revising [Mer09a], 2009-12-23.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2994.htm.

7 Revision history

Revision	Date	Changes
1.0	2013-03-12	• Published as N3545.