Tweaks to Streamline Concepts Lite Syntax

Document #: WG21 N4434
Date: 2015-04-10
Project: JTC1.22.32 Programming Language C++
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper proposes three small tweaks to simplify the syntax used by programmers in defining and using concepts as specified in [N4377].

1 Introduction

[N4377], on track to become a Technical Specification, provides wording to realize the long-awaited C++ language feature known as *concepts*. Based on the design proposed in [N3351], this extension provides the following new major language features:

- two forms of declarations for concepts (*function concept*s and *variable concept*s), each introduced by a new keyword, **`concept`**;
- a *requires-expression*, an unevaluated context introduced by a new **`requires`** keyword, used (typically within a concept definition) to express constraints on template arguments;
- a new *requires-clause*, also introduced by the **`requires`** keyword, to express constraints on template declarations and on function declarations; and
- *template-introduction*s, *constrained-parameter*s, and *constrained-type-specifier*s as more economical syntactic alternatives to *requires-clause*s.

These features have been implemented by Andrew Sutton (with assistance from his student Braden Obrzut) on gcc's **`c++-concepts`** branch [http://gcc.gnu.org/svn/gcc/branches/c++-concepts]. We have been extensively experimenting with this implementation. Based on our experience, we are proposing a few small tweaks in the interest of reducing the effort needed on the part of programmers to define and use concepts.

2 Redundant `bool`

Following an introductory **`concept`** keyword, the declaration of a *variable concept* has the same syntax as the declaration of a variable template, and the declaration of a *function concept* has the same syntax as the declaration of a function template. This means that the **`concept`** keyword

is always followed by **bool**, since concepts must always yield a truth value when applied. (See [dcl.spec.concept]/5.2, 6.1.)

It seems unnecessary for a programmer always to write both **concept** and **bool** for each declared concept. We propose that **concept** suffice by itself, and that the compiler implicitly supply **bool** as each variable concept's type and as each function concept's return type.

3 Single syntax

A *requires-expression* that imposes constraints previously defined in a concept **C** uses the syntax **requires C<T>** when **C** is a variable concept and uses the syntax **requires C<T>()** (note the extra parentheses) when **C** is a function concept. This requires programmers to be aware, for each concept, how it is defined. The form of a concept's definition seems to be an implementation detail, since any requirements can be coded in either concept form.

We believe that having two forms for concept definition is a historical artifact: At the time the Concepts Lite proposal was designed, variable templates had not yet been added to the language; when they were, concept definitions in that form were grafted onto the proposal, supplementing the existing function style. However, we believe it is unnecessary and a source of confusion to have both forms. We therefore propose to eliminate function concepts in order to simplify rules for both implementers and users alike.

For example, excising function concepts will simplify the following special rule for turning a *constrained-parameter* into a predicate constraint: "If **C** is [a] variable concept, then **E** is the *id-expression* **TT**. Otherwise, **C** is a function concept and **E** is the function call **TT()**" [temp.param, (10.3), cross-reference elided].

We chose to keep the variable concept style, as it has less syntactic noise (no **return;** no argument list, . . .) than function concepts. We also note that the variable concept style is the only style used in the Concept Design paper [N3351]. Our proposal thus corresponds to the vision of the intended design.

4 Concept evaluation anywhere

Given a concept **C**, how can a programmer determine (e.g., for testing purposes) whether a given argument (say, a type **T**) satisfies the concept's constraints? At the moment, this is only possible by writing **C<T>** within a *requires-clause* (or equivalent, such as a *constrained-parameter*).

To obtain a **bool** value reflecting whether the argument satisfies the constraint, a programmer must, in other contexts, write such boilerplate code as the following:

```
template< class T >
constexpr bool
  satisfies_C( )  { return false; }

template< C T >  // equivalent to requires C<T> for class T
constexpr bool
  satisfies_C( )  { return true; }
```

Then the programmer's code can invoke **satisfies_C<T>()** to obtain the desired **bool** result.

Such circumlocution seems extreme. We therefore propose to allow a concept name plus appropriate arguments (e.g., the simple **C<T>**) in any context where a **bool** value may reasonably

appear. The general availability of this construct will eliminate any need for its indirect invocation via such undesirable circumlocution as that shown above.

5 Acknowledgments

We greatly appreciate the thoughtful remarks received from the readers of this paper's early drafts. Thank you.

6 Bibliography

[N3351] B. Stroustrup and A. Sutton (eds.): "A Concept Design for the STL." ISO/IEC JTC1/SC22/WG21 document N3351 (post-Issaquah mailing), 2012-01-13. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3351.pdf.

[N4377] Andrew Sutton: "Programming Languages—C++ Extensions for Concepts." ISO/IEC JTC1/SC22/WG21 document N4377 (mid-Urbana/Lexena mailing), 2015-02-09. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4377.pdf.

7 Document history

Version	Date	Changes
1	2015-04-10	• Published as N4434.