Abstract

This paper proposes to lift restrictions, currently imposed by the Concepts-Lite Working Draft [N4553], on the contexts in which a requires-expression is allowed to appear.

1 Introduction

The Concepts-Lite Working Draft, [N4553], provides wording for several new C++ language features. In our opinion, chief in importance among them are the requires-clause and requires-expression, each introduced by the new requires keyword. This paper seeks to lift certain restrictions imposed by the current wording on the use of a requires-expression.

Our proposal is similar to one made in our earlier paper [N4434]. Among other “tweaks” to the then-current draft of Concepts-Lite, we had proposed “to allow a concept name plus appropriate arguments ... in any context where a bool value may reasonably appear.” In the year since, we have continued to conduct very extensive experimentation with all the Concept-Lite features as implemented for the forthcoming gcc6. Based on our application of these language features, we now believe it appropriate (and possibly even more important) to allow the analogous relaxation for a requires-expression, too.

2 Proposal

According to [N4553], “A requires-expression provides a concise way to express requirements on template arguments” [expr.prim.req]/1. We agree, but also believe there is even greater utility to such an expression, which is currently limited (by [expr.prim.req]/4) as to the contexts in which it is allowed to appear:

A requires-expression shall appear only within a concept definition (7.1.7), or within the requires-clause of a template-declaration (Clause 14) or function declaration (8.3.5).

We propose to lift this restriction and thereby to allow such a construct to appear in any context that permits a bool-valued expression.
In particular, easing the above-cited limitations will avoid such boilerplate circumlocutions as:

```cpp
template< class T, class U >
constexpr bool
is_assignable_v = false;
```

```cpp
template< class T, class U >
requires requires( T&& t, U&& u )
{ std::forward<T>(t) = std::forward<U>(u); }
constexpr bool
is_assignable_v<T, U> = true;
```

in favor of the far more straightforward:

```cpp
template< class T, class U >
constexpr bool
is_assignable_v = requires( T&& t, U&& u )
{ std::forward<T>(t) = std::forward<U>(u); };
```

We could write similar Concepts-Lite code today, but would need to phrase it as a (variable or function) concept. to do so. We believe that’s not good enough, for a concept can’t (yet) be evaluated outside a requires-clause or equivalent environment. We urge the adoption of this proposal to allow a requires-expression the maximum possible utility.

3 Proposed wording

Modify subclause 5.1.4 [expr.prim.req] of WG21 draft [N4553] as indicated:

4 A requires-expression shall appear only within a concept definition (7.1.7), or within the requires-clause of a template declaration (Clause 14) or function declaration (8.3.5). [ Example: . . . – end example ] [ Note: . . . – end note ]

4 Bibliography


5 Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Changes</th>
</tr>
</thead>
</table>
| 1       | 2016-02-12 | • Published as P0266R0.

---

¹The apparent reduplication of this keyword is not an error: the first occurrence introduces a requires-clause, while the second introduces a requires-expression. This proposal will likely reduce the need for such stuttering.