Declval

Howard Hinnant
Vicente J. Botet Escriba

Table of Contents

Overview

The motivation for `declval` was introduced in N2958: Moving Swap Forward. Here follows a rewording of this chapter.

With the provision of decltype, late-specified return types, and default template-arguments for function templates a new generation of SFINAE patterns will emerge to at least partially compensate the lack of concepts on the C++0x timescale. Using this technique, it is sometimes necessary to obtain an object of a known type in a non-using context, e.g. given the declaration

```
template<class T>
T&& declval(); // not used
```

as part of the function template declaration

```
template<class To, class From>
decltype(static_cast<To>(declval<From>())) convert(From&&);
```

or as part of a class template definition

```
template<class> class result_of;

template<class Fn, class... ArgTypes>
struct result_of<Fn(ArgTypes...)>
{
  typedef decltype(declval<Fn>()(declval<ArgTypes>()...)) type;
};
```

The role of the function template declval() is a transformation of a type T into a value without using or evaluating this function. The name is supposed to direct the reader's attention to the fact that the expression `declval<T>()` is an lvalue if and only if T is an lvalue-reference, otherwise an rvalue. To extend the domain of this function we can do a bit better by changing its declaration to

```
template<class T>
typename std::add_rvalue_reference<T>::type declval(); // not used
```

which ensures that we can also use cv void as template parameter. The careful reader might have noticed that `declval()` already exists under the name create() as part of the definition of the semantics of the type trait is_convertible in the C++0x standard.

The provision of a new library component that allows the production of values in unevaluated expressions is considered important to realize constrained templates in C++0x where concepts are not available. This extremely light-weight function is expected to be part of the daily tool-box of the C++0x programmer.

Reference

```
#include <boost/utility/declval.hpp>
```

```
namespace boost {

    template <typename T>
    typename add_rvalue_reference<T>::type declval() noexcept; // as unevaluated operand

}  // namespace boost
```

The library provides the function template declval to simplify the definition of expressions which occur as unevaluated operands.

```
template <typename T>
typename add_rvalue_reference<T>::type declval();
```

Remarks: If this function is used, the program is ill-formed.

Remarks: The template parameter T of declval may be an incomplete type.

Example:

```
template <class To, class From>
decltype(static_cast<To>(declval<From>())) convert(From&&);
```

Declares a function template convert which only participates in overloading if the type From can be explicitly converted to type To.

History

boost 1.50

Fixes:

- #6570 Adding noexcept to boost::declval.