std::remove_const with const references

Asked 9 years, 1 month ago Modified 29 days ago Viewed 12k times

admittedly /ədˈmɪtɪdli/ adv. 诚然，不可否认地
contrived /kənˈtraɪvd/adj. 人为的；做作的；不自然的

26

Why does `std::remove_const` not convert `const T&` to `T&`? This admittedly rather contrived example demonstrates my question:

```
#include <type_traits>

int main()
{
    int a = 42;
    std::remove_const<const int&>::type b(a);

    // This assertion fails
    static_assert(
        !std::is_same<decltype(b), const int&>::value,
        "Why did remove_const not remove const?"
    );

    return 0;
}
```

The above case is trivially easy to fix, so for context, imagine the following:

```
#include <iostream>


template <typename T>
struct Selector
{
    constexpr static const char* value = "default";
};

template <typename T>
struct Selector<T&>
{
    constexpr static const char* value = "reference";
};

template <typename T>
struct Selector<const T&>
{
    constexpr static const char* value = "constref";
};

int main()
{
    std::cout
        << Selector<typename std::remove_const<const int&>::type>::value
        << std::endl;

    return 0;
}
```

In the above example, I'd expect `reference` to be shown, rather than `constref`.

c++ templates c++11 std

Share Edit Follow Flag

edited Apr 8, 2013 at 19:28 — Andy Prowl — 120k ● 22 ● 373 ● 446

asked Apr 8, 2013 at 19:19 — dafrito — 324 ● 1 ● 3 ● 8

Remember, there's no such thing as a const reference, only a reference to const. – xaxxon Jul 14, 2016 at 9:19

```
const int *const p3 = p2; // 靠右的 const 是顶层 const，靠左的是底层 const
const int &r = ci;        // 用于声明引用的 const 都是底层 const
```

1 Answer

Sorted by: Highest score (default)

26

`std::remove_const` removes *top level* `const`-qualifications. In `const T&`, which is equivalent to `T const&`, the qualification is not top-level: in fact, it does not apply to the reference itself (that would be meaningless, because references are immutable by definition), but to the referenced type.

Table 52 in Paragraph 20.9.7.1 of the C++11 Standard specifies, regarding `std::remove_const`:

> The member typedef type shall name the same type as `T` except that any **top-level** const-qualifier has been removed. [*Example*: `remove_const<const volatile int>::type` evaluates to `volatile int`, whereas `remove_const<const int*>::type` evaluates to `const int*`. — *end example*]

In order to strip `const` away, you first have to apply `std::remove_reference`, *then* apply `std::remove_const`, and then (if desired) apply `std::add_lvalue_reference` (or whatever is appropriate in your case).

NOTE: As Xeo mentions in the comment, you may consider using an alias template such as `Unqualified` to perform the first two steps, i.e. strip away the reference, then strip away the `const`- (and `volatile`-) qualification.

Share Edit Follow Flag

edited May 1 at 9:37 — Matt Murphy — 42 ● 6

answered Apr 8, 2013 at 19:26 — Andy Prowl — 120k ● 22 ● 373 ● 446

1 The first two are often grouped together under an `Unqualified<T>` alias. – Xeo Apr 8, 2013 at 19:40

Ah, I understand now. Thanks a lot for the explanation. :) – dafrito Apr 8, 2013 at 20:24

@dafrito: Glad it helped :) – Andy Prowl Apr 8, 2013 at 20:25

Give an example here of how to combine them. – Ciro Santilli Путлер Капут 六四事 Sep 17, 2016 at 6:57