Pass same object as const and non-const reference

Asked 7 years, 8 months ago Modified 7 years, 8 months ago Viewed 1k times

5

The following code compiles with g++ v4.8.1 and outputs `45`, but is its compilation guaranteed based on the standard? Would other compilers complain?

```
#include <iostream>
#include <vector>

void test(const std::vector<int>& a, std::vector<int>& b) {
  b[0] = 45;
}

int main() {
  std::vector<int> v(1,0);
  test(v, v);
  std::cout << v[0] << std::endl;
}
```

I understand that there's nothing inherently wrong with the function definition, but when calling `test` with the same object, `v`, I somewhat expected a warning that I was passing a single object as both a `const` and non-`const` reference.

c++ constants

Share Edit Follow Flag edited Jul 28, 2014 at 16:48 asked Jul 28, 2014 at 16:20 glinka 313 ●3 ●11

Why do you expect a compiler to reject this? – Borgleader Jul 28, 2014 at 16:21

Perhaps it would spot that I pass the same object as both a `const` and non-`const` reference, which seems contradictory. – glinka Jul 28, 2014 at 16:23

3 @glinka: There's no contradiction. It's perfectly safe and sensible to make a `const` reference to a mutable object, preventing modification via that reference, so the language allows that. The converse, making a mutable reference to a `const` object, wouldn't be safe, so that's not allowed. – Mike Seymour Jul 28, 2014 at 16:24

Yes, I understand this; however, I feel the compiler might/should prevent me from making such contradictory claims about an object by passing `v` as both arguments to `test`. – glinka Jul 28, 2014 at 16:28

you make no claims and this is a common scenario. Btw, the compiler may want to warn that the variable `a` is defined but not used in `test()`. – Walter Jul 28, 2014 at 16:30

3 Answers

Sorted by: Highest score (default)

7

There is no problem becuase the compiler consideres these two parameters as different references. To understand the code consider the following example

```
int i = 10;
const int &cr = i;
int &r = i;

r = 20;

std::cout << cr << std::endl;
```

Share Edit Follow Flag answered Jul 28, 2014 at 16:29 Vlad from Moscow 261k ●19 ●169 ●294

You are the only one who's answered the question I was trying to ask, thank you. I believe you mean `int i` in the first line though. I cannot edit it. – glinka Jul 28, 2014 at 16:33

@glinka Thanks. It was a typo.:) – Vlad from Moscow Jul 28, 2014 at 16:33

@glinka: If your question is more specific than what's actually written in your post, then you should edit your post. – Benjamin Lindley Jul 28, 2014 at 16:40

1 I thought the title might suffice, but I can see where the confusion arose. Especially as a poster with a paltry 17 rep, you run into these "dumbifications" a lot. – glinka Jul 28, 2014 at 16:44

2

Yes, it is correct. A const reference can be bound to a non-const object. This is of course the case, because why would a non-const object care if a function *doesn't* modify it, which is what the const ensures? (*at least, it ensures that the object is not modified through that particular reference, though it may be modified through another, non-const reference*)

The reverse, however, is not true. You cannot bind a non-const reference to a const object. A const object does in fact care if a function modifies it, which a non-const reference would allow.

Share Edit Follow Flag edited Jul 28, 2014 at 16:28 answered Jul 28, 2014 at 16:22 Benjamin Lindley 98.4k ●9 ●188 ●265

+1 Your answer explains the behavior concisely, and precisely. – πάντα ῥεῖ Jul 28, 2014 at 16:29

2

There is no reason this shouldn't compile. Your vector is not const, you can then use it in a const or mutable context.

The same way this works :

```
int i = 42;
const int& const_ref = i;
int& ref = i;
```

Wether you bind the same object or not does not have any incidence here.

You should look at this as if there was a lion in a zoo, the visitors behind windows cannot touch him whereas the trainers can feed him, but it is still the same lion.

Share Edit Follow Flag edited Jul 28, 2014 at 16:26 answered Jul 28, 2014 at 16:21 Drax 11.9k ●6 ●39 ●79