Do c++11 lambdas capture variables they don't use?

When I use `[=]` to indicate that I would like all local variables to be captured by value in a lambda, will that result in *all* local variables in the function being copied, or just all local variables *that are used by the lambda*?

So, for example, if i I have:

```
vector<int> my_huge_vector(100000);
int my_measly_int;
some_function([=](int i){ return my_measly_int + i; });
```

Will my_huge_vector be copied, even though I don't use it in the lambda?

c++ lambda c++11

edited Feb 19 '14 at 23:16 — Drew Noakes — 105k 67 344 441

asked May 30 '11 at 23:02 — HighCommander4 — 13.8k 9 58 129

3 Answers

Each variable expressly named in the capture list is captured. The default capture will only capture variables that are both (a) not expressly named in the capture list and (b) *used* in the body of the lambda expression. If a variable is not expressly named and you don't use the variable in the lambda expression, then the variable is not captured. In your example, `my_huge_vector` is not captured.

Per C++11 §5.1.2[expr.prim.lambda]/11:

> If a *lambda-expression* has an associated *capture-default* and its *compound-statement odr-uses* `this` or a variable with automatic storage duration and the *odr-used* entity is not explicitly captured, then the *odr-used* entity is said to be implicitly captured.

Your lambda expression has an associated capture default: by default, you capture variables by value using the `[=]`.

If and only if a variable is used (in the One Definition Rule sense of the term "used") is a variable implicitly captured. Since you don't use `my_huge_vector` at all in the body (the "compound statement") of the lambda expression, it is not implicitly captured.

To continue with §5.1.2/14

> An entity is captured by copy if
>
> - it is implicitly captured and the *capture-default* is `=` or if
> - it is explicitly captured with a capture that does not include an `&`.

Since your `my_huge_vector` is not implicitly captured and it is not explicitly captured, it is not captured at all, by copy or by reference.

edited Oct 15 '12 at 20:23

answered May 30 '11 at 23:09 — James McNellis — 226k 36 656 814

6 Does thou hath a Holy Quote? – GManNickG May 30 '11 at 23:20

1 @GMan: I sure do! – James McNellis May 30 '11 at 23:26

I will say, though, that the entirety of §5.1.2 is important to understand all of the details. There are a lot of technical terms defined in that section and because the definitions of the various components of lambda expressions are necessarily entangled, it is difficult to pull out short quotes that definitively say "this is X and this is why X." – James McNellis May 30 '11 at 23:33

Pinging for your attention here, which says such optimization is not allowed, at least for explicitly named variables. I'm not sure where to draw the line. – GManNickG Oct 3 '12 at 23:54

@GManNickG: That is some mighty good trolling ;-). It took me a good three clicks of that link before I realized it actually did point to this page... :-O [In any case, I'll re-read the language spec when I get into the office tomorrow morning and update the answer appropriately.] – James McNellis Oct 4 '12 at 5:55

No, `my_huge_vector` will not be captured. `[=]` means all **used** variables are captured in the lambda.

answered May 30 '11 at 23:10

Thomas Minor
589 2 8

5 Yep. Note that *used* is a technical word, though, and really means the One Definition Rule *used*. So, for example, consider `void f() { const int size(10); [] { int x[size]; }; }`. Here, `size` is not captured but that's okay because it isn't used in the ODR sense. (Visual C++ 2010 does not accept this code, either because the spec changed after VC10 was released or due to a bug, presumably this will be fixed in a forthcoming version; g++ 4.5.1 accepts it.) – James McNellis May 31 '11 at 2:21

There's no guarantee either way. The compiler is allowed to copy your huge vector, even if you don't request a copy, just for giggles. OTOH if it can detect that the value is not being used, the compiler is allowed to elide the copy, even if explicitly requested.

Having said that, only variables that are used by the lambda are implicitly captured.

answered May 30 '11 at 23:15

n.m.
37.6k 2 46 101

The FDIS §5.1.2/11 explicitly says otherwise. – ildjarn May 30 '11 at 23:28

Well, I mean rather §12.2, probably not too seriously. The topicstarter did not mention a context where a compiler is allowed to create a temporary for the variable in question, so perhaps I was a tad too liberal in estimation of what could happen. – n.m. May 30 '11 at 23:54