Name Lookup Should “Find the First Thing of That Name”

Document #: WG21 P1753R0
Date: 2019–06–16
Audience: EWG ⇒ CWG
Reply to: Walter E. Brown <webrown.cpp@gmail.com>

Contents

Abstract

This paper observes that the state of C++ name lookup algorithms is somewhat inconsistent when compared with the conventional approaches used by most nested, block-structured programming languages. One specific adjustment to the C++ rules is proposed, and collaborators are invited to suggest and help wordsmith future adjustments to bring C++ more in line with the principle of finding the nearest name (rather than, as applied in several cases today, a name of a sought category).

Fate tried to conceal him by naming him Smith.
— Oliver Wendell Holmes, Jr.

If names are not correct, language will not be in accordance with the truth of things.
— Confucius

1 Introduction

In a 2019–06–09 CWG reflector posting, our WG21 colleague Davis Herring recently presented the following (lightly reformatted) code among other examples of C++ name lookup concerns:

```
struct S { using I = int; };
template<int> struct A : S { };
namespace N {
 int S,A;
 S::I    i;  // ::S::I, per [basic.lookup.qual]/1
 A<0>::I j;  // ??
}
```

I responded to the reflector on 2019–06–11 as follows:

> FWIW, I am (and have for a very long time been) terribly bothered by most C++ exceptions to the traditional name lookup rules used by the overwhelming majority of nested, block-structured programming languages. . . . .

I strongly prefer that the above declaration of **i** be ill-formed, and likewise the subsequent declaration of **j**:

- In the case of **i**’s declaration, I believe (contrary to current wording in [basic.lookup.qual]/1) that lookup for **S** should find **N::S** and not look further. (Equivalently, I firmly believe that the declaration of **N::S** should hide the declaration of **::S**.) Thus, the token sequence **S::I** would be ill-formed.
- For similar reasons, I believe that the declaration of **N::A** should hide that of **::A**. Thus, the token sequence **A<0>::I** in **j**’s declaration would become ill-formed, as the lookup of **A** finds an **int**, namely **N::A**, and not the hidden template **::A**.

In my universe, no program that exploits today’s [basic.lookup.qual]/1 name lookup loophole (see below) should ever pass code review. By requiring that readers keep extra, nonlocal context in mind while studying such code, the program violates the mental form of the "locality of reference" principle and makes it more difficult to reason about the code. (OTOH, in practice, I can’t recall seeing any program that deliberately exploits this exceptional rule.)

I respectfully but very strongly urge that we reconsider the putative benefits of this and our other exceptions to the usual rules of name lookup in nested, block-structured languages such as ours. Let’s give serious thought to excising as many of these exceptions as possible in the interests of improved clarity, simplicity, uniformity, and teachability, if nothing else.

In particular, let’s start by striking from [basic.lookup.qual]/1 the entire sentence specifying that ". . . lookup of the name preceding that :: considers only namespaces, types, and [some] templates . . . ." I believe that this exception to the customary nested, block-structured name lookup rules is an unnecessary wart that offers at best questionable significant benefits.

I am absolutely delighted that our colleague Davis seems to be undertaking what promises to be a comprehensive and holistic review of our current medley of rules re name lookup and related matters, thus presenting us with a timely opportunity to discuss such issues early in the C++23 time frame. Thank you, Davis.

2 Discussion

Several CWG participants responded favorably to the above informal proposal, in some cases offering additional insights and suggestions for future consideration. Here are some of these comments, lightly reformatted.

- Balog Pal: “I think it would be a step in the good direction.”
- Richard Smith: “I strongly agree from a philosophical point of view, though I’m a little worried that we’ll end up breaking a lot of code that is (likely accidentally) shadowing a type that is later used in a *nested-name-specifier*.
  - Comment on this part by Billy O’Neal: “This sounds like a thing compilers can have a warning for? . . . ”
  - Comment on this part by Gabriel Dos Reis: “I don’t know of any compiler doing that. But it would be really useful to have that. . . . ”

  “From a practical perspective, if we made this change we could also allow **non_type::** as a short-hand for **std::remove_reference<decltype(non_type)>::type::**, permitting things like:

```
std::map<Thing1, Thing2> my_map;
my_map::const_iterator i = my_map.begin();
  // note:  for const-correctness I want "::const_iterator" here,
  // not the "::iterator" that "auto" would give me.
```

"(And yes, I know we now have `std::as_const` and `map::cbegin`, but to me those seem like circumlocutions in this case.)"

- John Spicer: "My recollection of history could be a little off on this, but I think this originated from the 'struct hack'.

  "However, we have had a number of opportunities to reject 'normal' lookup rules and chosen not to. I'm not saying those were good decisions, I'm just saying that this isn't the only special case.

  "As an example, if you say `namespace X = Y::Z`, only namespaces are considered when looking up `Z`.

  "My preference would be to have all lookups simply find the first thing of that name. We would need to retain the C part of the struct hack.

  - Comment on this part by Gabriel Dos Reis: "Put me down for this."

  "There are other cases where we extended the struct hack where I think we should not have. For example, `using N::X` brings in both a type and nontype(s) in `N` where I would have preferred that it do the lookup and only bring in the one thing that resulted from a normal lookup.

  "I doubt we can get rid of the struct hack, but I think we can get rid of the C++ things that are similar to the struct hack but are not actually C.

  "The using declaration case would be one of those.

  "We could deprecate the special lookup for things that precede `::` so that compilers could warn on those.

  "I don't think there would be much code where accidental shadowing of a type used in a *nested-name-specifier* would occur, but I think it potentially enough that an initial deprecation step would be warranted."

We find considerable elegance in John Spicer's theme, "have all lookups simply find the first thing of that name" as opposed to current C++ rules that selectively ignore names attached to certain kinds of entities. Thus, this paper is an attempt to start an EWG conversation along these lines.

We provide, in the next section, one specific wording proposal, and would be pleased to collaborate with EWG colleagues in preparing additional proposals for further future name lookup simplifications.

3 Proposed wording[1]

3.1 Edit [basic.lookup.qual]/1 as shown:

The name of a class or namespace member or enumerator can be referred to after the `::` scope resolution operator (7.5.4.2) applied to a *nested-name-specifier* that denotes its class, namespace, or enumeration. If a `::` scope resolution operator in a *nested-name-specifier* is not preceded by a *decltype-specifier*, ~~lookup of~~ the name preceding that `::` ~~considers only namespaces, types, and templates whose specializations are types.If the name found does not~~ shall designate a namespace or a class, enumeration, or dependent type, else the program is ill-formed. [*Example*: . . .

[1]Proposed additions and ~~deletions~~ are based on [N4810]. Editorial instructions and drafting notes look like this.

4 Acknowledgments

Many thanks to the readers of early drafts of this paper for their thoughtful comments. Special thanks to John Spicer for inspiring this paper’s title.

5 Bibliography

[N4810] Richard Smith: “Working Draft, Standard for Programming Language C++.” ISO/IEC JTC1/SC22/WG21 document N4810 (post-Kona mailing), 2019–03–15. https://wg21.link/n4810.

6 Document history

Rev.	Date	Changes
0	2019–06–16	• Published as P1753R0, pre-Cologne mailing.