N4021: "Access Qualifiers" for C2y

Philipp Klaus Krause

November 18, 2026

We propose bringing the concept of "access qualifiers" from the Embedded C TR into the C2y standard. While all qualifiers except `_Atomic` in the C standard itself are "access qualifiers" having the concept is still useful, since many implementations have their own implementation-qualifiers (in particular qualifiers for intrinsic named address spaces from the Embedded C standard), and the term "access qualifiers" both make it easier to document how those behave and helps avoid breaking them when changing the C standard.

This proposal follows the earlier N3651, N3723, N3795, N3973 (Krause), and N3931, N3985 (Colomar).

Compared to most of those earlier proposals, this one focuses just on the "access qualifier" concept by itself instead of introducing it as part of a bigger change.

0 Motivation

Implementations often extend C via their own qualifiers. Those qualifiers behave differently, with many of them being quite different from `const` and `volatile`. For those similar to `const` and `volatile`, the Embedded C TR (see N1275 for a draft) introduced the term "access qualifier". In the C2y draft, `_Atomic` is currently the only qualifier that is not an access qualifier (but it also differs from qualifiers further, since atomic types differ from other types, and there is the `_Atomic` specifier, too). This means that there is a risk that when changing the C standard, we break qualifiers that are neither `_Atomic` not access qualifiers.

One important, and very widely implemented example of qualifiers that are not "access qualifiers" are those for intrinsic named address spaces from the Embedded C TR (see N3795 for a discussion of various implementations supporting them). In case we ever want to bring the named address spaces from the Embedded C TR into the main standard we will need the "access qualifier" concept. But having it already is helpful by itself, since it makes it substantially easier to maintain documentation for such qualifiers - both in the implementation's documentation and in future revision of the Embedded C TR. It also helps with documenting the behaviour of other implementation-defined qualifiers.

1 Introduce the "access qualifier" concept - proposed changes vs. N3886

1.1 6.2.5p38:

Change

"Each unqualified type has several *qualified versions* of its type,40) corresponding to the combinations of one, two, or all three of the `const`, `volatile`, and `restrict` qualifiers. The qualified or unqualified versions of a type are distinct types that belong to the same type category and have the same representation and alignment requirements.31) An array and its element type are always considered to be identically qualified.32) Any other derived type is not qualified by the qualifiers (if any) of the type from which it is derived."

to

"Each unqualified type has several *qualified versions* of its type,40) corresponding to the combinations of one, two, or all three of the `const`, `volatile`, and `restrict` qualifiers. There may also be implementation-defined qualifiers. The qualified or unqualified versions of a type are distinct types that belong to the same type category and have the same representation and alignment requirements.31) An array and its element type are always considered to be identically qualified.32) Any other derived type is not qualified by the qualifiers (if any) of the type from which it is derived. The qualifiers `const`, `volatile`, and `restrict` are *access qualifiers*. For each implementation-defined qualifier, it is implementation-defined whether it is an access qualifier."

1.2 6.2.6.3p1:

Change

"Similarly, pointers to qualified or unqualified versions of compatible types shall have the same representation and alignment requirements."

to

"Similarly, pointers to access-qualified or unqualified versions of compatible types shall have the same representation and alignment requirements."

1.3 6.3.3.3p2:

Change

"For any qualifier q, a pointer to a non-q-qualified type can be converted to a pointer to the q-qualified version of the type;"

to

"For any access qualifier q, a pointer to a non-q-qualified type can be converted to a pointer to the q-qualified version of the type."

1.4 6.5.1p9:

Change

"a qualified version of a type compatible with the effective type of the object"

to

"an access-qualified version of a type compatible with the effective type of the object"

and

"the signed or unsigned type corresponding to a type compatible with a qualified version of the underlying type of the effective type of the object,"

to

"the signed or unsigned type corresponding to a type compatible with a access-qualified version of the underlying type of the effective type of the object,"

1.5 6.5.7p3:

Change "both operands are pointers to qualified or unqualified versions of compatible complete object types;"

to

"both operands are pointers to access-qualified or unqualified versions of compatible complete object types;"

1.6 6.5.9p2:

Change "both operands are pointers to qualified or unqualified versions of compatible complete object types."

to

"both operands are pointers to access-qualified or unqualified versions of compatible complete object types."

1.7 6.5.10p2:

Change "both operands are pointers to qualified or unqualified versions of compatible complete object types;"

to

"both operands are pointers to access-qualified or unqualified versions of compatible complete object types;"

and

"one operand is a pointer to an object type and the other is a pointer to a qualified or unqualified version of void;"

to

"one operand is a pointer to an object type and the other is a pointer to an access-qualified or unqualified version of void;"

1.8 6.5.10p6:

Change "If one operand is a pointer to an object type and the other is a pointer to a qualified or unqualified version of void, the former is converted to the type of the latter."

to

"If one operand is a pointer to an object type and the other is a pointer to an access-qualified or unqualified version of void, the former is converted to the type of the latter."

1.9 6.5.16p3:

Change “both operands are pointers to qualified or unqualified versions of compatible types;”

to

“both operands are pointers to access-qualified or unqualified versions of compatible types;”

and

Change “one operand is a pointer to an object type and the other is a pointer to a qualified or unqualified version of void.”

to

“one operand is a pointer to an object type and the other is a pointer to a access-qualified or unqualified version of void.”

1.10 6.5.16p8:

Change “If both the second and third operands are pointers, the result type is a pointer to a type qualified with all the type qualifiers of the types referenced by both operands;”

to

“If both the second and third operands are pointers, the result type is a pointer to a type qualified with all the access qualifiers of the types referenced by both operands;”

1.11 6.5.17p2:

Change “If the value being stored in an object is read from another object that overlaps in any way the storage of the first object, then the two objects shall occupy exactly the same storage and shall have qualified or unqualified versions of a compatible type;”

to

“If the value being stored in an object is read from another object that overlaps in any way the storage of the first object, then the two objects shall occupy exactly the same storage and shall have access-qualified or unqualified versions of a compatible type;”

1.12 J3.11:

Add

“(3) Any implementation-defined qualifiers that exist in the implementation and which of them are access qualifiers(6.2.5).”

1.13 J5:

Add

“**J.5.20 Named address spaces**

Identifiers can denote address spaces. Objects are allocated in one or more *address spaces*. Named address spaces are supported via address space qualifiers (which are qualifiers, but not access qualifiers). The implementation documents

constraints and behaviour of additive, relational, equality, conditional, and assignment operators with operands that are are pointers to types qualified with address space qualifiers"

2 Further changes: library functions - proposed changes vs. N3886

The current wording for varying pointer arguments to some library functions is problematic wrt. pointers to qualified types. Adding named address space type qualifiers to the standard results in more problems, but the embedded C TR contains no wording to deal with the issue. Pointers to `_Atomic` or to a named address space are obviously problematic (since the standard does not require their representations to be compatible). But the same is true for other qualifiers, since these functions are not required to respect any `volatile` or `restrict`. The only qualifier that makes sense on the pointer target in some cases is `const`. The changes proposed here go further than just dealing with named address space type qualifiers, and disallow the use of pointers to further qualifiers.

2.1 7.24.6.2p8:

Change "the argument shall be a pointer to storage of character type."
to
"the argument shall be a pointer to storage of unqualified or `const`-qualified character type.",
and change
"the argument shall be a pointer to storage of `wchar_t` type"
to
"the argument shall be a pointer to storage of unqualified or `const`-qualified `wchar_t` type",
and
"The argument shall be a pointer to `void` or a pointer to a character type."
to
"The argument shall be a pointer to unqualified or access-qualified `void` or a pointer to an unqualified or access-qualified character type.",
and change
"The argument shall be a pointer to signed integer"
to
"The argument shall be a pointer to unqualified signed integer".

2.2 7.24.6.3p10:

Change
"If this object does not have an appropriate type"
to
"If this object does not have an appropriate unqualified type".

2.3 7.16.2.2p2:

Change all four (one of them is in a footnote)
"qualified or unqualified"
to
"access-qualified or unqualified".

2.4 7.28.5.1p3:

Change
"The `volatile` and `restrict` qualifiers are not accepted on the elements of the array to search."
to
"Qualifiers other than `const` are not accepted on the elements of the array to search."

2.5 7.33.2.2p8:

Same changes as 7.24.6.2p8.

2.6 7.33.2.3p10:

Same changes as 7.24.6.3p10.

2.7 7.33.4.6.1p3:

Same changes as 7.28.5.1p3.

3 Straw Polls

- Do we want the "access qualifier" concept in C2y using the wording above?
- Do we want to make the changes to the library functions using the wording above?