N2575

ISO/IEC 9899: Proposed enhancement for C2X
Allowing the programmer to define the type to be used to represent an enum

Introduction

This paper is an update of N2533, which itself was an update of N2008. N2533 was discussed during the August virtual meeting (Freiburg). In summary, it proposed:

1. Allowing the programmer to specify the type to be used to represent an enum. These 'new style' enums being referred to as 'enumeration with fixed underlying type', to distinguish them from traditional C-style enums 'enumeration without fixed underlying type'
2. During declaration, a member of an enumeration with fixed underlying type could only be specified with a value representable in that type (assuming infinite arithmetic, i.e. ignoring overflow and wraparound for the indicated type in the assumed abstract machine)
3. Only a value of the correct enum type may be assigned to a value of enumeration with fixed underlying type without an explicit cast
4. In all other way an enumeration with fixed underlying type behaves as an enumeration without fixed underlying type.

Note that these change don't impact legacy code, and are consistent with recent additions to C++. Also, Clang already supports this feature for C, so there is evidence of 'prior art'.

During the August virtual meeting there was:

- general support for adding 'enumerations with fixed underlying type'
- some disquiet about not using normal C integer arithmetic when initialising members
- there was discussion on the desirability of preventing the assignment of integers to a value of enumeration with fixed underlying type without an explicit cast.
  A vote was taken on "whether it was desirable to allow an implicit cast of integer to enum" (i.e. not the aim of the N2533). The results were yes=4 no=7 abstain=8, i.e. a small majority in favour of the N2533 proposal in this area.
  However, in discussion after the meeting it was suggested that there may not be an easy way to achieve this, as the idea that an enum is an integer is widely dispersed in the standard, and modifying behaviour in some places but not others may lead to contractions in the standard

A number of more detailed comments and questions were discussed

1. *the proposal says the underlying type must be an integer, enums and _Bool are integers, are they allowed*? That wasn't the intention.
2. *the proposal says const & volatile qualifiers on the underlying type are ignored, what about _Atomic?* _Atomic should be ignored as well
3. *are attributes allowed on the type?* They should be
4. *a macro or similar to get the underlying type from an enum would be useful*

5. *How does an enumeration with fixed underlying type work with a generic selection – are the enumeration with fixed underlying type and the underlying type treated as different types or the same*?

The aim of this update is to address the discussion by restricting the proposed changes to just those required to allow the programmer to define the integer type used to represent an enum, and taking in the first three comments above. I'd agree that bullet 4 would be useful, but is essentially a separate issue. Bullet 5 is a good question, but as there already seems to be a complex relationship between enumeration types and the integer type the compiler chooses to represent them with, with respect to generic selection, I don't believe this proposal makes the position worse.

A future proposal may revisit the prevention of assigning an integer to an enum without a cast.

Required changes to current working draft N2478

6.4.4.3 Enumeration constants

Syntax

enumeration-constant:

identifier

Semantics

An identifier declared as an enumeration constant for an enumeration without fixed underlying type has type **int**.

An identifier declared as an enumeration constant for an enumeration with fixed underlying type has that underlying type during the specification of the enumeration type (i.e. until the closing brace in *enum-specifier)*. When used in an integer context, it is treated as if it had the underlying type.

Forward references: enumeration specifiers (6.7.2.2).

6.7.2.2 Enumeration specifiers

Syntax[1]

enum-specifier:

enum *attribute-specifier-sequence*$_{opt}$ *identifier*$_{opt}$ *enum-type-specifier*$_{opt}$ ***{*** *enumerator-list* ***}***

[1] For drafting – some of the syntax lines may wrap

enum *attribute-specifier-sequence*$_{opt}$ *identifier*$_{opt}$ *enum-type-specifier*$_{opt}$ ***{*** *enumerator-list* ***, }***

enum *identifier* *enum-type-specifier*$_{opt}$

enumerator-list:

enumerator

enumerator-list *,* *enumerator*

enumerator:

enumeration-constant *attribute-specifier-sequence*$_{opt}$

enumeration-constant *attribute-specifier-sequence*$_{opt}$ **=** *constant-expression*

enum-type-specifier:

: *specifier-qualifier-list*

All enumerations have an <new term>underlying type</new term>. The underlying type can be explicitly specified using an *enum-type-specifier* and such an underlying type is said to be <new term>fixed</new term>.

Constraints

The type specifiers in the *specifier-qualifier-list* shall specify an integer type that is not an enumerated type or _Bool. No alignment specifiers shall appear in the *specifier-qualifier-list*. The underlying type of the enumeration is the unqualified, non-atomic version of the type specified by the type specifiers in the specifier-qualifier-list.

For an enumeration without a fixed underlying type, the underlying type of the enumeration is **int.**~~The expression that defines the value of an enumeration constant shall be an integer constant expression that has a value representable as an **int**.~~

An enumeration with a fixed underlying type may be redeclared, provided both declarations have *enum-type-specifier*s that resolve to the same integer type.

Semantics

The optional attribute specifier sequence in the **enum** specifier appertains to the enumeration; the attributes in that attribute specifier sequence are thereafter considered attributes of the enumeration whenever it is named. The optional attribute specifier sequence in the enumerator appertains to that enumerator.

The identifiers in an enumerator list are declared as constants that have the underlying type of the enumeration,~~that have type **int**~~), and may appear wherever such are permitted.[133]

An enumerator with **=** defines its enumeration constant as the value of the constant expression. If the first enumerator has no **=,** the value of its enumeration constant is 0. Each subsequent enumerator with no **=** defines its enumeration constant as the value of the constant expression obtained by adding 1 to the value of the previous enumeration constant. (The use of enumerators with **=** may produce enumeration constants with values that duplicate other values in the same enumeration.) The enumerators of an enumeration are also known as its members.

For all enumerations without fixed underlying type, e~~E~~ach enumerated type shall be compatible with **char**, a signed integer type, or an unsigned integer type. The choice of type is implementation-defined,[134] but shall be capable of representing the values of all the members of the enumeration.

For an enumeration with a fixed underlying type, the enumerated type shall be compatible with the underlying type of the enumeration.

An~~The~~ enumerated type declaration with an *enumerator-list* is an incomplete type until immediately after the **}** that terminates the list of enumerator declarations, and complete thereafter. A declaration of an enumeration with fixed underlying type without an *enumerator-list* declares a complete type.

EXAMPLE The following fragment:

< *unchanged* + new example>

```
        enum E1: short;                         /* Forward reference */
        enum E2: short;

        enum E1: short { m11, m12 };
        enum E2: long  { m21, m22 };   /* Constraint violation */

         enum E1 x = m11;     /* enum-type-specifier not needed */
```

6.7.2.3 Tags
Constraints

< paras 1 & 2 unchanged >

3 A type specifier of the form
enum *identifier*
without an enumerator list shall only appear after ~~the type it specifies is complete~~its underlying type is determined. For an enum without fixed underlying type, this is after the closing '}' of the enumerator list.

< paras 4..6 unchanged >

7 A type specifier of the form

struct-or-union attribute-specifier-sequenceopt identifier$_{opt}$ **{** *member-declaration-list* **}**
or
enum *attribute-specifier-sequenceopt identifier*$_{opt}$ *enum-type-specifier*$_{opt}$ **{** *enumerator-list* **}**
or
enum *attribute-specifier-sequenceopt identifier*$_{opt}$ *enum-type-specifier*$_{opt}$ **{** *enumerator-list* **, }**

declares a structure, union, or enumerated type. The list defines the structure content, union content, or enumeration content. If an identifier is provided,[137]) the type specifier also declares the identifier to be the tag of that type. The optional attribute specifier sequence appertains to the structure, union, or enumeration type being declared; the attributes in that attribute specifier sequence are thereafter considered attributes of the structure, union, or enumeration type whenever it is named.

Acknowledgements:

Thanks to Aaron Ballman for help with the initial drafting, and Aaron, Jens Gustedt & Joseph Myers for questions, suggestions and offline discussion.