N2533

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 N2008, originally discussed in Spring 2016 (London). At that point it was accepted in principle for C2X, but with comments.

The main comment was that my rather clumsy concrete syntax for *enum-type-specifier* was overly restrictive. It wouldn't for example allow the use of a typedef. This has been fixed by using appropriate syntax terms from the current working draft, but with a constraint. The syntax has also been changed to allow for attributes.

The original proposal was for two changes to be made to the specification and use of enums. The first proposal was independent of the second, but the second required the first to have been adopted. However, as one of the motivations for these changes is compatibility with C++, and C++ requires both changes, this document now doesn't separate them.

Neither change impacts legacy code, and the changes are consistent with recent additions to C++. Also, Clang already supports this feature for C, so there is evidence of 'prior art'.

Note: there is a need to distinguish between C17 style enums (still available after these changes) and the new, C++ compatible, enums. The terminology 'enumeration without fixed underlying type' (old style) and 'enumeration with fixed underlying type' (C++ style) is used to distinguish them.

Outline

This proposal allows the programmer to (optionally) define the integer type to be used to represent an enum, as in: enum E1: unsigned int {……}; with the semantics that the compiler will always use a member of the indicated type to represent an object of type enum E1.

When declaring enumeration constants for an enumeration with fixed underlying type , it is a constraint error if the associated value cannot be represented in the indicated type, whether the value is explicit or implicit:

```
enum E2 : unsigned char
          { m1 =  -1,  /* constraint error, not unsigned */
            m2 = 255,
            m3         /* constraint error, 256 not in the
                          range of unsigned char         */
```

```
};
```

A variable with a type that is an enumeration with fixed underlying type may only be initialised or assigned a member of the same enumeration type, without an explicit cast. This is for increased type safety and compatibility with C++.

Enumeration constants of an enumeration with fixed underlying type may be used in expressions as integers of the underlying type.

There is no change to the behaviour of enumeration without fixed underlying type.

```
enum E1               { m11,  m12, m13}; /* old style enum     */
enum E2: unsigned int { m21,  m22, m23}; /* new style enum     */

enum E1 a = m11;                         /* legal – no change  */
enum E1 b = 2;                           /* legal – no change  */
enum E1 c = m11 | m12;                   /* legal – no change  */
        c = 2;                           /* legal – no change  */
        c = m11 | m12;                   /* legal – no change  */
int     x = m11 | m12;                   /* legal – no change  */
        x = m11 | m12;                   /* legal – no change  */

enum E2 d = m21;                         /* legal              */
enum E2 e = 2;                           /* constraint error   */
enum E2 f = m21 | m22;                   /* constraint error   */
enum E2 g = (enum E2)2;                  /* legal              */
enum E2 h = (enum E2)(m11 | m12);        /* legal              */
        h = m21 | m22;                   /* constraint error   */
        h = (enum E2)(m21 | m22);        /* legal              */
 int     y = m21 | m22;                   /* legal              */
        y = m21 | m22;                   /* legal              */
```

Rationale

This proposal came from people working on embedded systems, and on the MISRA C working group. It is suggested that this would confer the following advantages:

1. Improved type safety, by making explicit the type to be used to represent an enum
2. The ability to match with hardware registers and packet data structure definitions in terms of the width assigned to enumerated field values.
3. Improved and explicit control over the amount of storage allocated to an enum object.
4. Be compatible with C++ enums
5. This feature helps the stability of ABIs for library developers

6. The removal of the perceived ambiguity (and portability issues) of the integer type of an enum and hence the need for type casts when moving to and from other integer types when the design intent is that they have compatible storage.

Status

This paper is about 80% complete.

The following sections include detailed editing instructions for: the specification of enumeration types and constants, and initialisation of enumeration type objects, in the current draft standard.

What is not currently addressed are the wording changes needed for assignment to enumeration type variables. Aaron and I were uncertain where they should go, so decided to submit the paper in its current state and invite feedback and suggestions from the committee.

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 integral 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* ***}***

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*

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

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

For an enumeration with a fixed underlying type, the *specifier-qualifier-list* in an enum type specifier shall resolve to an integral type after substitution of any **typedefs** and ignoring any const or volatile type qualifiers. The underlying type of the enumeration is the adjusted type specified.

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**.~~

The expression that defines the value of an enumeration constant, shall be representable by the underlying type of the enumeration.

An enumerations with 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;
enum E2: short;

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

Notes

There is a potential parsing ambiguity, for example:

```
struct S {
        enum e : 12;  /*  is it an unnamed bit-field of incomplete enum type
                          or an erroneous enum-type-specifier?                */
    };
```

A : following "enum *attribute-specifier-sequence*$_{opt}$ *identifier*$_{opt}$" within a *member-declaration* of a *member-declaration-list* is parsed as part of an *enum-specifier*.

6.7.9 Initialization

...

Constraints

{new constraint} If the type of the entity to be initialized is an enumeration with fixed underlying type, then it shall be initialized with either a value of that enumeration type or an enumerator constant of that enumeration type.

Semantics

{new} Example NN an object with type of an enumeration with fixed underlying type can only be initialised with a member of the same type, without an explicit cast

```
enum E3 : unsigned int { m31, m32, m33 };
enum E3 a = m32;
enum E3 b = (enum E3)42;
```

There needs to be a similar constraint added for assignment – not sure where that should go.