**N2008**

# ISO/IEC 9899: Proposed enhancement for C2X

## Allowing the programmer to define the type to be used to represent an enum

Outline

This note proposes two changes to be made to the specification and use of enums. The second proposal requires the first to have been adopted, but may be ignored without impacting the first.

Proposal 1: programmer definition of the type representing an enum

This proposal is to allow 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.

Syntax

The current syntax for an enum-specifier is:

*enum-specifier:*
    **enum** *identifier*$_{opt}$ **{** *enumerator-list* **}**
    **enum** *identifier*$_{opt}$ **{** *enumerator-list* **,** **}**
    **enum** *identifier*

The proposed syntax is:

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

where

*enum-type-specifier:*

    **:** *type-specifier*$_{opt}$ *type-specifier*$_{opt}$ *type-specifier*$_{opt}$ *type-specifier*

with a constraint that the only valid combinations are:

**char**
**short, short int**
**int**
**long, long int**
**long long** or **long long int**

optionally preceded by **unsigned** or **signed**, or **unsigned** or **signed** on their own.

It shall also be a constraint error if the *enumerator-list* attempts to define an enum member with a value that cannot be represented in the indicated type, as in:

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

Rationale

This proposal has come from people working on embedded systems, and on the MISRA 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. 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.

This proposal is a subset of that already adopted by WG21 for C++11.

## Proposal 2: remove the implicit cast from integral types to (new style) enums

It is also suggested that type safety can be further improved by removing the implicit cast from integer types to enums, as in the following.

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

enum E1 a = m11;                         /* legal   */
enum E1 b = 2;                           /* legal   */
enum E1 c = m11 | m12;                   /* legal   */

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

- This only applies to enums declared as described in proposal 1 (as otherwise this would be too disruptive to existing code)
- Enum values can still be implicitly used as integer values in expressions, procedure calls etc.
- This behaviour is compatible with that of C++

Clive Pygott, LDRA Inc. 17/2/2016