The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

6.4.4.3 Enumeration constants

enumeration-constant: 863
identifier

Commentary

There is no phase of translation where an identifier that is an *enumeration-constant* is replaced by its constant value.

C++

The C++ syntax uses the terminator *enumerator*.

Other Languages

Other languages that contain enumeration constants also specify them as identifiers.

Coding Guidelines

enumeration constant naming conventions

The issue of naming conventions for enumeration constant identifiers is discussed elsewhere.

Semantics

enumeration constant type

An identifier declared as an enumeration constant has type **int**. 864

Commentary

There is no requirement that the enumerated type containing this enumeration constant also have type **int**, although any constant expression used to specify the value of an enumeration constant is required to be representable in an **int**.

enumeration constant representable in int

C++

7.2p4 *Following the closing brace of an enum-specifier, each enumerator has the type of its enumeration. Prior to the closing brace, the type of each enumerator is the type of its initializing value. If an initializer is specified for an enumerator, the initializing value has the same type as the expression. If no initializer is specified for the first enumerator, the type is an unspecified integral type. Otherwise the type is the same as the type of the initializing value of the preceding enumerator unless the incremented value is not representable in that type, in which case the type is an unspecified integral type sufficient to contain the incremented value.*

This is quite a complex set of rules. The most interesting consequence of them is that each enumerator, in the same type definition, can have a different type (at least while the enumeration is being defined):
In C the type of the enumerator is always **int**. In C++ it can vary, on an enumerator by enumerator basis, for the same type definition. This behavior difference is only visible outside of the definition if an initializing value is calculated by applying the **sizeof** operator to a prior enumerator in the current definition.

```
#include <limits.h>

enum TAG { E1 = 2L,             // E1 has type long
           E2 = sizeof(E1),     // E2 has type size_t, value sizeof(long)
           E3 = 9,              // E3 has type int
           E4 = '4',            // E4 has type char
           E5 = INT_MAX,        // E5 has type int
           E6,                  // is E6 an unsigned int, or a long?
           E7 = sizeof(E4),     // E2 has type size_t, value sizeof(char)
         }                      // final type is decided when the } is encountered
           e_val;

int probably_a_C_translator(void)
{
return (E2 == E7);
}
```

Source developed using a C++ translator may contain enumeration with values that would cause a constraint violation if processed by a C translator.

```
1  #include <limits.h>
2
3  enum TAG { E1 = LONG_MAX }; /* Constraint violation if LONG_MAX != INT_MAX */
```

Other Languages

In most other languages that contain enumeration constants, the type of the enumeration constant is the enumerated type, a type that is usually different from the integer types.

Common Implementations

Some implementations support the use of the type **`short`** and **`long`**, in the enumeration declaration,[3] to explicitly specify the type of the enumeration constant, while others[2] use pragmas, or command line options.[1]

Coding Guidelines

The potential uses for enumeration constants are discussed elsewhere. On the whole these uses imply that an enumeration constant belongs to a unique type— the enumerated type it is defined within. Treating an enumeration constant as having some integer type only becomes necessary when use is made of its value—for instance, as an operand in an arithmetic or bitwise operation. The issues involved in mixing objects having enumerated type and the associated enumeration constants of that type as operands in an expressions are discussed elsewhere.

enumeration
set of named constants

enumeration
set of named constants

865 **Forward references:** enumeration specifiers (6.7.2.2).

References

1. Diab Data. *D-CC & D-C++ Compiler Suites User's Guide*. Diab Data, Inc, www.ddi.com, 4.3 edition, June 1999.
2. Metroworks. *CodeWarrior C Compilers Reference*. Metroworks Corp., Aug. 2001.
3. H. Packard. *HP C/HP-UX Programmer's Guide*. Hewlett Packard Company, tenth edition, Dec. 2001.