Strongly Typed, Scoped Enumerations

An **enum class** is an alternative enumeration type that provides simultaneously (1) an enclosing scope for its enumerators and (2) stronger typing compared to a classic **enum**.

Description

C++11 introduces a novel enumeration construct, **enum class** or, equivalently, **enum struct**:

```
enum class  Ec { A, B, C }; // scoped enumeration, Ec, containing three enumerators
```

The enumerators of the **enum class** Ec in the line above — namely, A, B, and C — do not automatically become part of the enclosing scope and must be qualified to be referenced:

```
Ec e0 = A;      // Error, A not found
Ec e1 = Ec::A;  // OK
```

Moreover, attempting to use an expression of type **enum class** E as, say, an **int** or in an arithmetic context will be flagged as an error, thus necessitating an explicit cast:

```
int  i0 = Ec::B;                    // Error, conversion to int not supported
int  i1 = static_cast<int>(Ec::B);  // OK, i1 is 1.
int  i2 = 1 + Ec::B;                // Error, conversion to int not supported
int  i3 = -Ec::B;                   // Error, unsupported arithmetic operations

bool b0 = Ec::B != 2;               // Error, comparison with int unsupported
bool b1 = Ec::B != Ec::C;           // OK, b1 is 'true'.
```

The **enum class** *complements* but does not replace the classical, C-style **enum**:

```
enum E { e_Enumerator0 /*= value0 */, /*...*/ e_EnumeratorN /* = valueN */ };
    // Classic, C-style enum: enumerators are neither type safe nor scoped.
```

For examples where the classic **enum** shines, see *Potential Pitfalls — Strong typing of an **enum class** can be counterproductive* on page 344 and *Annoyances — Scoped enumerations do not necessarily add value* on page 351.

Still, innumerable practical situations occur in which enumerators that are both scoped and more type safe would be preferred; see *Introducing the C++11 scoped enumerations* on page 335 and *Use Cases* on page 337.

Drawbacks and workarounds relating to unscoped C++03 enumerations

Since the enumerators of a classic **enum** leak out into the enclosing scope, if two unrelated enumerations that happen to use the same enumerator name appear in the same scope, ~~an ambiguity~~ could ensue: