```
    clearScreen(Color::e_RED, 1           );  // Error, no matching function
    clearScreen(Color::e_RED, 1.0         );  // Error, no matching function
    clearScreen(Color::e_RED, Color::e_RED);  // Error, no matching function
}
```

Bottom line: Having a *pure* enumeration be strongly typed — such as `Color`, used widely in function signatures — can help to expose accidental misuse but, again, see *Potential Pitfalls — Strong typing of an* ***enum class*** *can be counterproductive* on page 344.

Note that strongly typed enumerations help to avoid accidental misuse by requiring an explicit cast should conversion to an arithmetic type be desired:

```
void f6()
{
    clearScreen(Color::e_RED, 1.0);                   // Error, no match
    clearScreen(static_cast<int>(Color::e_RED), 1.0); // OK, calls (2) above
    clearScreen(Color::e_RED, 1.0, false);            // OK, calls (5) above
}
```

Encapsulating implementation details within the enumerators themselves

In rare cases, providing a pure, ordered enumeration having unique (but not necessarily contiguous) numerical values that exploit lower-order bits to categorize and make readily available important individual properties might offer an advantage, such as in performance. Note that to preserve the ordinality of the enumerators overall, the higher-level bits must encode their relative order. The lower-level bits are then available for arbitrary use in the implementation.

For example, suppose that we have a `MonthOfYear` enumeration that encodes in the least-significant bit the months that have 31 days and an accompanying `inline` function to quickly determine whether a given enumerator represents such a month:

```
#include <type_traits>  // std::underlying_type

enum class MonthOfYear : unsigned char  // optimized to flag long months
{
    e_JAN = ( 1 << 4) + 0x1,
    e_FEB = ( 2 << 4) + 0x0,
    e_MAR = ( 3 << 4) + 0x1,
    e_APR = ( 4 << 4) + 0x0,
    e_MAY = ( 5 << 4) + 0x1,
    e_JUN = ( 6 << 4) + 0x0,
    e_JUL = ( 7 << 4) + 0x1,
    e_AUG = ( 8 << 4) + 0x1,
    e_SEP = ( 9 << 4) + 0x0,
```