```
int hoursToSeconds0(int hours)
    // Return the number of seconds in the specified hours.  The behavior is
    // undefined unless the result can be represented as an int.
{
    return hours * 3600;
}
```

This use of *magic constants* has, however, long been known[4] to make finding uses of the constants and the relationships between related ones needlessly difficult. For integral values only, we could always represent such compile-time constants symbolically by using a classic **enum** type, in deliberate preference to the modern, type-safe enumerator; see Section 2.1."**enum class**" on page 332:

```
struct TimeRatios1  // explicit scope for single classic anonymous enum type
{
    enum  // anonymous enumeration comprising related symbolic constants
    {
        k_SECONDS_PER_MINUTE = 60,      // Underlying type (UT) might be int.
        k_MINUTES_PER_HOUR   = 60,
        k_SECONDS_PER_HOUR   = 60*60  // These enumerators have the same UT.
    };
};

int hoursToSeconds1(int hours)
    // ...
{
    return hours * TimeRatios1::k_SECONDS_PER_HOUR;
}
```

This traditional solution, while often effective, gives little control over the underlying integral type of the enumerator used to represent the symbolic compile-time constant, leaving it at the mercy of the totality of values used to initialize the members of the enumeration. Such inflexibility might lead to compiler warnings and nonintuitive behavior resulting from **enum**-specific integral promotion rules, especially when the **underlying type (UT)** used to represent the time ratios differs from the integral types with which they are ultimately used; see Section 2.1."Underlying Type '11" on page 829.

In this particular example, extending the **enum** to cover ratios up to a week and conversions down to microseconds would manifestly change its UT (because there are far more than 2^{32} microseconds in a week), altering how all of the enumerators behave when used in expressions with, say, values of type **int**:

```
struct TimeRatios2  // explicit scope for single classic anonymous enum type
{
    enum  // Anonymous enumeration --- UT is governed by all of the enumerators.
    {
```

[4]**kernighan99**, section 1.5, pp. 19–22