```
        k_SECONDS_PER_MINUTE = 60,    // UT might be long or long long.
        k_MINUTES_PER_HOUR   = 60,
        k_SECONDS_PER_HOUR   = 60*60,
        // ...
        k_USEC_PER_WEEK = 1000L*1000*60*60*24*7  // same UT as all of the above
    };
};
```

The original *values* will remain unchanged after the enumeration is extended, but the burden of all of the compiler warnings resulting from the change in UT and rippling throughout a large codebase could be expensive to repair.

We would like the original values to remain unchanged (e.g., remain as **int** if that's what they were), and we want only those values that do *not* fit in an **int** to morph into a larger integral type. We might achieve this effect by placing each enumerator in its own separate anonymous enumeration:

```
struct TimeRatios3  // explicit scope for multiple classic anonymous enum types
{
    enum { k_SECONDS_PER_MINUTE = 60               };  // UT: int (likely)
    enum { k_MINUTES_PER_HOUR   = 60               };  //  "   "      "
    enum { k_SECONDS_PER_HOUR   = 60*60            };  //  "   "      "
    // ...
    enum { k_USEC_PER_SEC  = 1000*1000             };  // UT: int (likely)
    enum { k_USEC_PER_MIN  = 1000*1000*60          };  //  "   "      "
    enum { k_USEC_PER_HOUR = 1000U*1000*60*60      };  // UT: unsigned int
    enum { k_USEC_PER_DAY  = 1000L*1000*60*60*24   };  // UT: long or long long
    enum { k_USEC_PER_WEEK = 1000L*1000*60*60*24*7 };  // UT: long or long long
};
```

In this case, the original values as well as their respective UTs will remain unchanged, and each new enumerated value will independently take on its own independent UT, which is ~~either implementation defined or else dictated~~ by the number of bits required to represent the value.

A modern alternative to having separate anonymous **enum**s for each distinct value (or class of values) is to instead encode each ratio as an explicitly typed **constexpr** variable:

```
struct TimeRatios4
{
    static constexpr int k_SECONDS_PER_MINUTE      = 60;
    static constexpr int k_MINUTES_PER_HOUR        = 60;
    static constexpr int k_SECONDS_PER_HOUR        = k_MINUTES_PER_HOUR *
                                                     k_SECONDS_PER_MINUTE;
    // ...
    static constexpr long long k_NANOS_PER_SECOND  = 1000*1000*1000;
    static constexpr long long k_NANOS_PER_HOUR    = k_NANOS_PER_SECOND *
                                                     k_SECONDS_PER_HOUR;
};
```