digit to the left of the binary point. When a number gets small enough, however, it's stored in a different format in which it's not normalized. These ***subnormal numbers*** (also known as ***denormalized numbers*** or ***denormals***) can be much smaller than normalized numbers; the trade-off is that they get progressively less accurate as they get smaller.

- *Special values.* Each floating-point format allows the representation of three special values: ***positive infinity, negative infinity,*** and *NaN* ("not a number"). Dividing a positive number by zero produces positive infinity. Dividing a negative number by zero yields negative infinity. The result of a mathematically undefined operation, such as dividing zero by zero, is NaN. (It's more accurate to say "the result is *a* NaN" rather than "the result is NaN," because the IEEE standard has multiple representations for NaN. The exponent part of a NaN value is all 1 bits, but the fraction can be any nonzero sequence of bits.) Special values can be operands in subsequent operations. Infinity behaves just as it does in ordinary mathematics. For example, dividing a positive number by positive infinity yields zero. (Note that an arithmetic expression could produce infinity as an intermediate result but have a noninfinite value overall.) Performing any operation on NaN gives NaN as the result.
- *Rounding direction.* When a number can't be stored exactly using a floating-point representation, the current ***rounding direction*** (or ***rounding mode***) determines which floating-point value will be selected to represent the number. There are four rounding directions: (1) *Round toward nearest.* Rounds to the nearest representable value. If a number falls halfway between two values, it is rounded to the "even" value (the one whose least significant bit is zero). (2) *Round toward zero.* (3) *Round toward positive infinity.* (4) *Round toward negative infinity.* The default rounding direction is round toward nearest.
- *Exceptions.* There are five types of floating-point exceptions: *overflow, underflow, division by zero, invalid operation* (the result of an arithmetic operation was NaN), and *inexact* (the result of an arithmetic operation had to be rounded). When one of these conditions is detected, we say that the exception is ***raised.***

Types

C99 adds two types, `float_t` and `double_t`, to `<math.h>`. The `float_t` type is at least as "wide" as the `float` type (meaning that it could be the `float` type or any wider type, such as `double`). Similarly, `double_t` is required to be at least as wide as the `double` type. (It must also be at least as wide as `float_t`.) These types are provided for the programmer who's trying to maximize the performance of floating-point arithmetic. `float_t` should be the most efficient floating-point type that's at least as wide as `float`; `double_t` should be the most efficient floating-point type that's at least as wide as `double`.

FLT_EVAL_METHOD ➤*23.1*

The `float_t` and `double_t` types are related to the `FLT_EVAL_METHOD` macro, as shown in Table 23.8.