Table 3: Available precisions for various IEEE-754 floating-point types

Name	Common Name	Significant Bits[a]	Decimal Digits	Exponent Bits	Dynamic Range
binary16	Half precision	11	3.31	5	$\sim 6.5 \times 10^5$
binary32	Single precision	24	7.22	8	$\sim 3.4 \times 10^{38}$
binary64	Double precision	53	15.95	11	$\sim 10^{308}$
binary80	Extended precision	69	20.77	11	$\sim 10^{308}$
binary128	Quadruple precision	113	34.02	15	$\sim 10^{4932}$

[a] Note that the most significant bit of the **mantissa** is always a `1` for normalized numbers and 0 for denormalized ones and, hence, is not stored explicitly. Thus one additional bit remains to represent the sign of the overall floating-point value; the sign of the exponent is encoded using **excess-n** notation.

Determining the minimum number of decimal digits needed to accurately approximate a transcendental value, such as π, for a given type on a given platform can be tricky and require some **binary-search**-like detective work, which is likely why overshooting the precision without warning is the default on most platforms. One way to establish that *all* of the decimal digits in a given floating-point literal are relevant for a given floating-point type is to compare that literal and a similar one with its least significant decimal digit removed[6]:

```
static_assert(3.1415926535f != 3.141592653f, "too precise for float");
    // This assert will fire on a typical platform.

static_assert(3.141592653f != 3.14159265f, "too precise for float");
    // This assert too will fire on a typical platform.

static_assert(3.14159265f != 3.1415926f, "too precise for float");
    // This assert will not fire on a typical platform.

static_assert(3.1415926f != 3.141592f, "too precise for float");
    // This assert too will not fire on a typical platform.
```

If the values are *not* the same, then that floating-point type can make use of the precision suggested by the original literal; if they *are* the same, however, then it is likely that the available precision has been exceeded. Iterative use of this technique by developers can help them to empirically narrow down the maximal number of decimal digits a particular platform

[6] Note that affixing the `f` (literal *suffix*) to a floating-point literal is equivalent to applying a **`static_cast<float>`** to the (unsuffixed) literal:

```
static_assert(3.14'159'265'358f == static_cast<float>(3.14'159'265'358),"");
```