Appendix

Historical perspective on the evolution of use of fundamental integral types

The designers of C got it right back in 1972 when they created a portable **`int`** type that could act as a bridge from a single-word, 16-bit, integer, **`short`**, to a double-word, 32-bit, integer, **`long`**. Just by using **`int`**, one would get the optimal space versus speed trade-off as the 32-bit computer *word* was on its way to becoming the norm. As an example, the Motorola 68000 series (c. 1979) was a hybrid CISC architecture employing a 32-bit instruction set with 32-bit registers and a 32-bit external data bus; internally, however, it used only 16-bit ALUs and a 16-bit data bus.

During the late 1980s and into the 1990s, the word size of the machine and the size of an **`int`** were synonymous. Some of the earlier mainframe computers, such as IBM 701 (c. 1954), had a word size of 36 ~~characters~~ (1) to allow accurate representation of a signed 10-digit decimal number or (2) to hold up to six 6-bit characters. Smaller computers, such as Digital Equipment Corporation's PDP-1, PDP-9, and PDP-15 used 18-bit words, so a double word held 36 bits; memory addressing, however, was limited to just 12–18 bits, i.e., a maximum 4K–256K 18-bit words of DRAM. With the standardization of 7-bit ASCII (c. 1967), its adoption throughout the 1970s, and its last update (c. 1986), the common typical notion of character size moved from 6 to 7 bits. Some early conforming implementations of C would choose to set `CHAR_BIT` to 9 to allow two characters per half word. (On some early vector-processing computers, `CHAR_BIT` is 32, making every type, including a **`char`**, at least a 32-bit quantity.) As double-precision floating-point calculations — enabled by type **`double`** and supported by floating-point coprocessors — became typical in the scientific community, machine architectures naturally evolved from 9-, 18-, and 36-bit words to the familiar 8-, 16-, 32-, and now 64-bit addressable integer words we have today. Apart from embedded systems and digital signal processors, a **`char`** is now almost universally considered to be exactly 8 bits. Instead of scrupulously and actively using `CHAR_BIT` for the number of bits in a **`char`**, consider statically asserting it instead:

```
static_assert(CHAR_BIT == 8, "A char is not 8-bits on this CrAzY platform!");
```

As cost of *main memory* was decreasing exponentially throughout the final two decades of the 20th century,[5] the need for a much larger *virtual address space* quickly followed. Intel began its work on 64-bit architectures in the early 1990s and realized one a decade later. As we progressed into the 2000s, the common notion of word size, i.e., the width ~~(in bits)~~ of typical registers within the CPU itself, began to shift from "the ~~size~~ of an **`int`**" to "the ~~size~~ of a simple (nonmember) pointer type," e.g., `8 * sizeof(void*)`, on the host platform. By this time, 16-bit **`int`** types — like 16-bit architectures for **general-purpose machines**, i.e., excluding **embedded systems** — were long gone, but a **`long int`** was still expected to be 32 bits on a 32-bit platform. Embedded systems are designed specifically to work with high-performance hardware, such as digital-signal processors. Sadly, **`long`** was often

[5]Moore's law (c. 1965) — the observation that the number of transistors in densely packed integrated circuits (e.g., DRAM) grows exponentially over time, doubling every 1–2 years or so — held for nearly a half century, until finally saturating in the 2010s.