<limits.h> header ➤23.2

(These macros augment the ones in the <limits.h> header.) <stdint.h> also defines parameterized macros that construct integer constants with specific types. There are no functions in <stdint.h>.

The primary motivation for the <stdint.h> header lies in an observation made in Section 7.5, which discussed the role of type definitions in making programs portable. For example, if i is an int variable, the assignment

```
i = 100000;
```

is fine if int is a 32-bit type but will fail if int is a 16-bit type. The problem is that the C standard doesn't specify exactly how many bits an int value has. The standard *does* guarantee that the values of the int type must include all numbers between –32767 and +32767 (which requires at least 16 bits), but that's all it has to say on the matter. In the case of the variable i, which needs to be able to store 100000, the traditional solution is to declare i to be of some type T, where T is a type name created using typedef. The declaration of T can then be adjusted based on the sizes of integers in a particular implementation. (On a 16-bit machine, T would need to be long int, but on a 32-bit machine, it can be int.) This is the strategy that Section 7.5 discusses.

If your compiler supports C99, there's a better technique. The <stdint.h> header declares names for types based on the ***width*** of the type (the number of bits used to store values of the type, including the sign bit, if any). The typedef names declared in <stdint.h> may refer to basic types (such as int, unsigned int, and long int) or to extended integer types that are supported by a particular implementation.

sign bit ➤7.1

<stdint.h> Types

The types declared in <stdint.h> fall into five groups:

- ***Exact-width integer types.*** Each name of the form int*N*_t represents a signed integer type with *N* bits, stored in two's-complement form. (Two's complement, a technique used to represent signed integers in binary, is nearly universal among modern computers.) For example, a value of type int16_t would be a 16-bit signed integer. A name of the form uint*N*_t represents an unsigned integer type with *N* bits. An implementation is required to provide both int*N*_t and uint*N*_t for *N* = 8, 16, 32, and 64 if it supports integers with these widths.

- ***Minimum-width integer types.*** Each name of the form int_least*N*_t represents a signed integer type with at least *N* bits. A name of the form uint_least*N*_t represents an unsigned integer type with *N* or more bits. <stdint.h> is required to provide at least the following minimum-width types:

  ```
  int_least8_t     uint_least8_t
  int_least16_t    uint_least16_t
  ```