```
typedef long int ptrdiff_t;
typedef unsigned long int size_t;
typedef int wchar_t;
```

C99

<stdint.h> header ➤27.1

In C99, the `<stdint.h>` header uses `typedef` to define names for integer types with a particular number of bits. For example, `int32_t` is a signed integer type with exactly 32 bits. Using these types is an effective way to make programs more portable.

7.6 The `sizeof` Operator

The `sizeof` operator allows a program to determine how much memory is required to store values of a particular type. The value of the expression

`sizeof` expression

`sizeof ( `*type-name*` )`

is an unsigned integer representing the number of bytes required to store a value belonging to *type-name*. `sizeof(char)` is always 1, but the sizes of the other types may vary. On a 32-bit machine, `sizeof(int)` is normally 4. Note that `sizeof` is a rather unusual operator, since the compiler itself can usually determine the value of a `sizeof` expression.

Q&A

The `sizeof` operator can also be applied to constants, variables, and expressions in general. If `i` and `j` are `int` variables, then `sizeof(i)` is 4 on a 32-bit machine, as is `sizeof(i + j)`. When applied to an expression—as opposed to a type—`sizeof` doesn't require parentheses; we could write `sizeof i` instead of `sizeof(i)`. However, parentheses may be needed anyway because of operator precedence. The compiler would interpret `sizeof i + j` as `(sizeof i) + j`, because `sizeof`—a unary operator—takes precedence over the binary + operator. To avoid problems, I always use parentheses in `sizeof` expressions.

Printing a `sizeof` value requires care, because the type of a `sizeof` expression is an implementation-defined type named `size_t`. In C89, it's best to convert the value of the expression to a known type before printing it. `size_t` is guaranteed to be an unsigned integer type, so it's safest to cast a `sizeof` expression to `unsigned long` (the largest of C89's unsigned types) and then print it using the `%lu` conversion:

```
printf("Size of int: %lu\n", (unsigned long) sizeof(int));
```

C99

In C99, the `size_t` type can be larger than `unsigned long`. However, the `printf` function in C99 is capable of displaying `size_t` values directly, without needing a cast. The trick is to use the letter `z` in the conversion specification, followed by one of the usual integer codes (typically `u`):

```
printf("Size of int: %zu\n", sizeof(int));   /* C99 only */
```