pointer parameter. but it always has `int *` type.) Having a type-generic macro for `modf` would pose some difficult problems. For example. the meaning of `modf (d, &f)`, where d has type `double` and f has type `float`, is unclear: are we calling the `modf` function or the `modff` function? Rather than develop a complicated set of rules for a single function (and probably taking into account that `modf` isn't a very popular function), the C99 committee chose not to provide a type-generic `modf` macro.

Q: When a `<tgmath.h>` macro is invoked with an integer argument, the `double` version of the corresponding function is called. Shouldn't the `float` version be called, according to the usual arithmetic conversions? [p. 725]

A: We're dealing with a macro. not a function. so the usual arithmetic conversions don't come into play. The C99 committee had to create a rule for determining which version of a function would be called when an integer argument is passed to a `<tgmath.h>` macro. Although the committee at one point considered having the `float` version called (for consistency with the usual arithmetic conversions), they eventually decided that choosing the `double` version was better. First, it's safer: converting an integer to `float` may cause a loss of accuracy, especially for integer types whose width is 32 bits or more. Second, it causes fewer surprises for the programmer. Suppose that i is an integer variable. If the `<tgmath.h>` header isn't included. the call `sin(i)` calls the `sin` function. On the other hand, if `<tgmath.h>` *is* included, the call `sin(i)` invokes the `sin` macro; because i is an integer. the preprocessor replaces the `sin` macro with the `sin` function, and the end result is the same.

Q: When a program invokes one of the type-generic macros in `<tgmath.h>`, how does the implementation determine which function to call? Is there a way for a macro to test the types of its arguments?

A: One unusual aspect of `<tgmath.h>` is that its macros need to be able to test the types of the arguments that are passed to them. C has no features for testing types, so it would normally be impossible to write such a macro. The `<tgmath.h>` macros rely on special facilities provided by a particular compiler to make such testing possible. We don't know what these facilities are, and they're not guaranteed to be portable from one compiler to another.

Exercises

Section 27.1

1. (C99) Locate the declarations of the `intN_t` and `uintN_t` types in the `<stdint.h>` header installed on your system. Which values of *N* are supported?

2. (C99) Write the parameterized macros `INT32_C(n)`, `UINT32_C(n)`, `INT64_C(n)`, and `UINT64_C(n)`. assuming that the `int` type and `long int` types are 32 bits wide and the `long long int` type is 64 bits wide. *Hint:* Use the `##` preprocessor operator to attach