Q&A Moreover, assigning a value to a variable of a narrower type will give a meaningless result (or worse) if the value is outside the range of the variable's type:

```
c = 10000;    /*** WRONG ***/
i = 1.0e20;   /*** WRONG ***/
f = 1.0e100;  /*** WRONG ***/
```

A "narrowing" assignment may elicit a warning from the compiler or from tools such as `lint`.

It's a good idea to append the `f` suffix to a floating-point constant if it will be assigned to a `float` variable, as we've been doing since Chapter 2:

```
f = 3.14159f;
```

Without the suffix, the constant `3.14159` would have type `double`, possibly causing a warning message.

C99 Implicit Conversions in C99

`_Bool` type ➤5.2

The rules for implicit conversions in C99 are somewhat different from the rules in C89, primarily because C99 has additional types (`_Bool`, `long long` types, extended integer types, and complex types).

For the purpose of defining conversion rules, C99 gives each integer type an "integer conversion rank." Here are the ranks from highest to lowest:

1. `long long int`, `unsigned long long int`
2. `long int`, `unsigned long int`
3. `int`, `unsigned int`
4. `short int`, `unsigned short int`
5. `char`, `signed char`, `unsigned char`
6. `_Bool`

For simplicity, I'm ignoring extended integer types and enumerated types.

In place of C89's integral promotions, C99 has "integer promotions," which involve converting any type whose rank is less than `int` and `unsigned int` to `int` (provided that all values of the type can be represented using `int`) or else to `unsigned int`.

As in C89, the C99 rules for performing the usual arithmetic conversions can be divided into two cases:

- *The type of either operand is a floating type.* As long as neither operand has a complex type, the rules are the same as before. (The conversion rules for complex types will be discussed in Section 27.3.)
- *Neither operand type is a floating type.* First perform integer promotion on both operands. If the types of the two operands are now the same, the process ends. Otherwise, use the following rules, stopping at the first one that applies:
  - If both operands have signed types or both have unsigned types, convert the