- Signed variables can be maddening and the source of frustration as far as creepy errors are concerned. It works like this: Suppose that you add 1 to a signed integer variable. If that variable already holds the value `32,767`, its new value (after you add 1) is `-32,768`. Yes, even though you *add* a number, the result is negative. In that instance, you should be using an `unsigned int` variable type to avoid the problem.
- To use an unsigned variable and skirt around the negative-number issue, you must declare your variables by using either the `unsigned int` or `unsigned long` keyword. Your C compiler may have a secret switch that allows you to always create programs by using unsigned variables; refer to the online documentation to see what it is.

How to Make a Number Float

Two scoops of ice cream. . . .

Integer variables are the workhorses in your programs, handling most of the numeric tasks. However, when you have to deal with fractions, numbers that have a decimal part, or very large values, you need a different type of numeric variable. That variable is the *float*.

The `float` keyword is used to set aside space for a variable designed to contain a floating-point, or noninteger, value. Here's the format:

```
float var;
```

The keyword `float` is followed by a space or a tab, and then comes the variable name, *var*. The line ends in a semicolon.

Or, you can declare a `float` variable and give it a value, just as you can any other variable in C:

```
float var=value;
```

In this format, the variable *var* is followed by an equal sign and then a *value* to be assigned to it.

Float is short for floating point. That term somehow refers to the decimal point in the number. For example, the following number is a floating-point value:

```
123.4567
```

An integer variable wouldn't cut it for this number. It could be only 123 or 124. When you have a decimal, you need a floating-point variable.