The range for floating-point numbers is quite large. With most C compilers, you can store any number in the range $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$. In English, that's a value between negative 340 undecillion and positive 340 undecillion. An undecillion is a 1 with 36 zeroes after it. That's a true, Mr. Spock-size value, though most numbers you use as floats are far less.

- Rules for naming variables are in Chapter 8.
- Noninteger values are stored in `float` variables.
- Even though 123 is an integer value, you can still store it in a `float` variable. However. . . .
- `float` variables should be used only when you need them. They require more internal storage and more PC processing time and power than integers do. If you can get by with an integer, use that type of variable instead.

"Hey, Carl, let's write a floating-point number program!"

Suppose that you and I are these huge, bulbous-headed creatures, all slimy and green and from the planet Redmond. We fly our UFO all over the galaxy, drink blue beer, and program in C on our computers. I'm Dan. Your name is Carl.

One day, while assaulting cows in Indiana, we get into this debate:

Dan: A light-year is 5,878,000,000,000 miles long! That's 5 trillion, 878 billion, plus change! I'm not walking that!

Carl: Nay, but it's only a scant 483,400,000 miles from the sun to Jupiter. That is but a fraction of a light-year.

Dan: How much of a fraction?

Carl: Well, why don't you type the following C program and have your computer calculate the distance for you?

Dan: Wait. I'm the author of this book. *You* type the program, JUPITER.C, and *you* figure it out. Sheesh.

```
#include <stdio.h>

int main()
{
    float lightyear=5.878E12;
    float jupiter=483400000;
    float distance;
```