```
    float x, y, z;
    x = 123456789.0;
    y = x + 1.23456789;
    z = y - x;
    printf("x=%.9f, y=%.9f, z=%.9f\n", x, y, z);
```

```
x=123456792.000000000, y=123456792.000000000, z=0.000000000
```

Somehow it is not possible to add small values to large numbers, and a cheap calculator is more accurate. You need to remember that fact when you are planning programs. Floating point arithmetic is risky, especially if the large values in question are later reduced to small values by further operations. Accumulated rounding errors have the potential to make any answer that you compute meaningless.

Variables of type `float` store approximate values over a much larger range than do `int` variables.

If additional precision is required, C offers a third numeric data type: the `double`. Variables of type `double` require more memory in the computer than do those of type `int` or `float`, but unless they are being used in very large arrays (Chapter 7), the overhead is small. Use of `double` variables reduces the potential for errors, but does not eliminate it. The minus sign on the first `z` output by this next program fragment indicates that the value computed is still non-zero, and rounding errors are again apparent in the low order digits of the `z` in the second output line. Like `float` values, `double` variables are printed using a "`%f`" format control:

```
    double x, y, z;
    x = 0.1;
    y = x+x+x+x+x+x+x+x+x+x;
    z = y - 1.0;
    printf("x=%.14f, y=%.14f, z=%.14f\n", x, y, z);
    x = 123456789.0;
    y = x + 1.23456789;
    z = y - x;
    printf("x=%.9f, y=%.9f, z=%.9f\n", x, y, z);
```

```
x=0.10000000000000, y=1.00000000000000, z=-0.00000000000000
x=123456789.000000000, y=123456790.234567896, z=1.234567896
```

Variables of type `double` provide increased accuracy in numeric computations. Even so, care should be taken that rounding errors do not affect the usefulness of computed values.

Constants also have types. All of the following are numeric constants:

```
    1                               -22
    345                             0
    3.14159                         10000.
    -.12345                         1e10
    -2.72e+0                        1.0e-6
    156e+52                         -2.78e-17
```

The "e" that appears in some of the numbers is an *exponent part*, and represents a scale factor as a power of ten. For example, `2.4e5` represents the number $2.4 \times 10^5 = 240{,}000$, and `1e-6` represents $1.0 \times 10^{-6} = 0.000001$.

Any numeric value that contains neither a decimal point nor an exponent part is inferred to be of type `int`. Numeric values that contain either a decimal point or an exponent are of type `double`. No `float` constants are created by the C compiler, a fact that can be verified by looking at the output produced by this program fragment:

```
    float x;
    x = 0.1;
    printf("x=%.20f, 0.1=%.20f\n", x, 0.1);
```

```
x=0.10000000149011611938, 0.1=0.10000000000000000555
```

It is also possible to explicitly declare numeric constants in several other ways, including as octal and hexadecimal integers. These options are beyond the scope of this book.

Use of inappropriate constants can create problems. For example, to implement the physics calculation $e = \frac{1}{2}mv^2$ to work out the kinetic energy e of a mass m moving at velocity v, it is tempting to write:

```
    float energy, mass, velocity;
    /* BEWARE -- incorrect code */
    energy = (1/2)*mass*velocity*velocity;
```

The problem is that in this form both `1` and `2` are integers, so an integer division is performed, and an initial result of integer zero obtained. Only when the multiplication by `mass` is considered is the integer value in the first subexpression converted to a `float`. By then it is too late – zero as an `int` converts to zero as a `float`. The fix is simple once the mistake is noticed – the factor `(1/2)` can be replaced by `0.5`, for example. But until the mistake is identified, the program is erroneous. Notice also that the C compiler is completely silent about the potential problem here. You have to be aware of this kind of thing as you write the program. The next section considers expressions, and type conversions during the evaluation of expressions, in more detail.

Be sure that you use constants of types that match the variables they are being combined with.

Character constants can also be created. Single quotes are used to delineate them: `'a'`, `'8'`, `' '`, and so on. Because the backslash character and quote character have special meanings, they are *escaped* to make character constants: `'\\'` and `'\''` respectively. Other special characters are newline, `'\n'`; and tab, `'\t'`. The corresponding variables are declared as type `char`.