```
modf(3.14159, &int_part)  ⇒  0.14159 (int_part is assigned 3.0)
```

Although `int_part` must have type `double`, we can always cast it to `int` or `long int` later.

frexp The `frexp` function splits a floating-point number into a fractional part f and an exponent n in such a way that the original number equals $f \times 2^n$, where either $0.5 \leq f < 1$ or $f = 0$. `frexp` returns f and stores n in the (integer) object pointed to by the second argument:

```
frexp(12.0, &exp)  ⇒  .75 (exp is assigned 4)
frexp(0.25, &exp)  ⇒  0.5 (exp is assigned -1)
```

ldexp `ldexp` undoes the work of `frexp` by combining a fraction and an exponent into a single number:

```
ldexp(.75, 4)    ⇒  12.0
ldexp(0.5, -1)   ⇒  0.25
```

In general, the call `ldexp(x, exp)` returns $x \times 2^{exp}$.

The `modf`, `frexp`, and `ldexp` functions are primarily used by other functions in `<math.h>`. They are rarely called directly by programs.

Power Functions

```
double pow(double x, double y);
double sqrt(double x);
```

pow The `pow` function raises its first argument to the power specified by its second argument:

```
pow(3.0, 2.0)   ⇒  9.0
pow(3.0, 0.5)   ⇒  1.73205
pow(3.0, -3.0)  ⇒  0.037037
```

sqrt `sqrt` computes the square root:

```
sqrt(3.0)  ⇒  1.73205
```

Using `sqrt` to find square roots is preferable to calling `pow`, since `sqrt` is usually a much faster function.

Nearest Integer, Absolute Value, and Remainder Functions

```
double ceil(double x);
double fabs(double x);
double floor(double x);
double fmod(double x, double y);
```