four directions, and so on. You might think that on average the drunkard doesn't move very far because the choices cancel each other out, but that is not the case.

Represent locations as integer pairs (x, y). Implement the drunkard's walk over 100 intersections, starting at (0, 0), and print the ending location.

■ **P6.8** A simple random generator is obtained by the formula

$$r_{\text{new}} = (a \cdot r_{\text{old}} + b)\%m$$

and then setting r_{old} to r_{new}. If m is chosen as 2^{32}, then you can compute

$$r_{\text{new}} = a \cdot r_{\text{old}} + b$$

because the truncation of an overflowing result to the `int` type is equivalent to computing the remainder.

Write a program that asks the user to enter a value for r_{old}. (Such a value is often called a *seed*). Then print the first 100 random integers generated by this formula, using $a = 32310901$ and $b = 1729$.

■■ **P6.9** *The Buffon Needle Experiment*. The following experiment was devised by Comte Georges-Louis Leclerc de Buffon (1707–1788), a French naturalist. A needle of length 1 inch is dropped onto paper that is ruled with lines 2 inches apart. If the needle drops onto a line, we count it as a *hit*. (See Figure 10.) Buffon discovered that the quotient *tries/hits* approximates π.

For the Buffon needle experiment, you must generate two random numbers: one to describe the starting position and one to describe the angle of the needle with the x-axis. Then you need to test whether the needle touches a grid line.

Generate the *lower* point of the needle. Its x-coordinate is irrelevant, and you may assume its y-coordinate y_{low} to be any random number between 0 and 2. The angle α between the needle and the x-axis can be any value between 0 degrees and 180 degrees (π radians). The upper end of the needle has y-coordinate

$$y_{\text{high}} = y_{\text{low}} + \sin\alpha$$

The needle is a hit if y_{high} is at least 2, as shown in Figure 11. Stop after 10,000 tries and print the quotient *tries/hits*. (This program is not suitable for computing the value of π. You need π in the computation of the angle.)

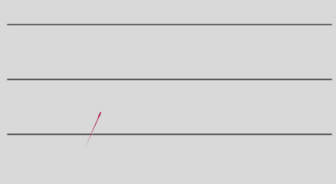

Figure 10
The Buffon Needle Experiment

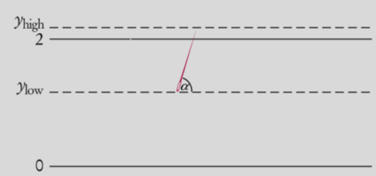

Figure 11
A Hit in the Buffon Needle Experiment

■■ **P6.10** In the 17th century, the discipline of probability theory got its start when a gambler asked a mathematician friend to explain some observations about dice games. Why did he, on average, lose a bet that at least one six would appear when rolling a die four times? And why did he seem to win a similar bet, getting at least one double-six when rolling a pair of dice 24 times?

6.10

Nowadays, it seems astounding that any person would roll a pair of dice 24 times in a row, and then repeat that many times over. Let's do that experiment on a computer instead. Simulate each game a million times and print out the wins and losses, assuming each bet was for $1.

© Charles Gibson/iStockphoto.

Science P6.14 In a predator-prey simulation, you compute the populations of predators and prey, using the following equations:

$$prey_{n+1} = prey_n \times (1 + A - B \times pred_n)$$
$$pred_{n+1} = pred_n \times (1 - C + D \times prey_n)$$

Here, A is the rate at which prey birth exceeds natural death, B is the rate of predation, C is the rate at which predator deaths exceed births without food, and D represents predator increase in the presence of food.

Write a program that prompts users for these rates, the initial population sizes, and the number of periods. Then print the populations for the given number of periods. As inputs, try $A = 0.1$, $B = C = 0.01$, and $D = 0.00002$ with initial prey and predator populations of 1,000 and 20.

an obviously dangerous experiment, so we will do it in the safety of the computer.

In fact, we will confirm the theorem from calculus by a simulation. In our simulation, we will consider how the ball moves in very short time intervals Δt. In a short time interval the velocity v is nearly constant, and we can compute the distance the ball moves as $\Delta s = v\Delta t$. In our program, we will simply set

© MOF/iStockphoto.

```
const double DELTA_T = 0.01;
```

▪▪ Science P6.15 *Projectile flight*. Suppose a cannonball is propelled straight into the air with a starting velocity v_0. Any calculus book will state that the position of the ball after t seconds is $s(t) = -\frac{1}{2}gt^2 + v_0 t$, where $g = 9.81\ \mathrm{m/s^2}$ is the gravitational force of the earth. No calculus textbook ever states why someone would want to carry out such an obviously dangerous experiment, so we will do it in the safety of the computer.

In fact, we will confirm the theorem from calculus by a simulation. In our simulation, we will consider how the ball moves in very short time intervals Δt. In a short time interval the velocity v is nearly constant, and we can compute the distance the ball moves as $\Delta s = v\Delta t$. In our program, we will simply set

© MOF/iStockphoto.

```
const double DELTA_T = 0.01;
```

302 Chapter 6 Loops

6.15

and update the position by

```
s = s + v * DELTA_T;
```

The velocity changes constantly—in fact, it is reduced by the gravitational force of the earth. In a short time interval, $\Delta v = -g\Delta t$, we must keep the velocity updated as

```
v = v - g * DELTA_T;
```

In the next iteration the new velocity is used to update the distance.

Now run the simulation until the cannonball falls back to the earth. Get the initial velocity as an input (100 m/s is a good value). Update the position and velocity 100 times per second, but print out the position only every full second. Also printout the values from the exact formula $s(t) = -\frac{1}{2}gt^2 + v_0 t$ for comparison.

Note: You may wonder whether there is a benefit to this simulation when an exact formula is available. Well, the formula from the calculus book is *not* exact. Actually, the gravitational force diminishes the farther the cannonball is away from the surface of the earth. This complicates the algebra sufficiently that it is not possible to give an exact formula for the actual motion, but the computer simulation can simply be extended to apply a variable gravitational force. For cannonballs, the calculus-book formula is actually good enough, but computers are necessary to compute accurate trajectories for higher-flying objects such as ballistic missiles.

304 Chapter 6 Loops

▪▪▪ Graphics P6.22 Draw a picture of the "four-leaved rose" whose equation in polar coordinates is $r = \cos(2\theta)$. Let θ go from 0 to 2π in 100 steps. Each time, compute r and then compute the (x, y) coordinates from the polar coordinates by using the formula

$$x = r \cdot \cos(\theta),\ y = r \cdot \sin(\theta)$$

R7.5 Write code that fills an array `values` with each set of numbers below.

a. 1	2	3	4	5	6	7	8	9	10	
b. 0	2	4	6	8	10	12	14	16	18	20
c. 1	4	9	16	25	36	49	64	81	100	
d. 0	0	0	0	0	0	0	0	0	0	
e. 1	4	9	16	9	7	4	9	11		
f. 0	1	0	1	0	1	0	1	0	1	
g. 0	1	2	3	4	0	1	2	3	4	

R7.6 Consider the following array:

```
int[] a = { 1, 2, 3, 4, 5, 4, 3, 2, 1, 0 };
```

What is the value of `total` after the following loops complete?

a.
```
int total = 0;
for (int i = 0; i < 10; i++) { total = total + a[i]; }
```
b.
```
int total = 0;
for (int i = 0; i < 10; i = i + 2) { total = total + a[i]; }
```
c.
```
int total = 0;
for (int i = 1; i < 10; i = i + 2) { total = total + a[i]; }
```
d.
```
int total = 0;
for (int i = 2; i <= 10; i++) { total = total + a[i]; }
```
e.
```
int total = 0;
for (int i = 1; i < 10; i = 2 * i) { total = total + a[i]; }
```
f.
```
int total = 0;
for (int i = 9; i >= 0; i--) { total = total + a[i]; }
```
g.
```
int total = 0;
for (int i = 9; i >= 0; i = i - 2) { total = total + a[i]; }
```
h.
```
int total = 0;
for (int i = 0; i < 10; i++) { total = a[i] - total; }
```

Review Exercises 359

R7.11 Write enhanced `for` loops for the following tasks.

a. Printing all elements of an array in a single row, separated by spaces.

b. Computing the maximum of all elements in an array.

c. Counting how many elements in an array are negative.

R7.12 Rewrite the following loops without using the enhanced `for` loop construct. Here, `values` is an array of floating-point numbers.

a.
```
for (double x : values) { total = total + x; }
```
b.
```
for (double x : values) { if (x == target) { return true; } }
```
c.
```
int i = 0;
for (double x : values) { values[i] = 2 * x; i++; }
```

R7.13 Rewrite the following loops using the enhanced `for` loop construct. Here, `values` is an array of floating-point numbers.

a.
```
for (int i = 0; i < values.length; i++) { total = total + values[i]; }
```
b.
```
for (int i = 1; i < values.length; i++) { total = total + values[i]; }
```
c.
```
for (int i = 0; i < values.length; i++)
{
   if (values[i] == target) { return i; }
}
```

■■ **R7.22** Suppose `values` is a *sorted* array of integers. Give pseudocode that describes how a new value can be inserted so that the resulting array stays sorted.

■■■ **R7.23** A *run* is a sequence of adjacent repeated values. Give pseudocode for computing the length of the longest run in an array. For example, the longest run in the array with elements

```
1 2 5 5 3 1 2 4 3 2 2 2 2 3 6 5 5 6 3 1
```

has length 4.