Let's construct a function that displays the contents of a type polar structure. The math functions in the C++ library assume angles are in radians, so we measure angles in that unit. But for display purposes, we convert radian measure to degrees. That means multiplying by 180/p, which is approximately 57.29577951. Here's the function:

```cpp
// show polar coordinates, converting angle to degrees
void show_polar (polar dapos)
{
    const double Rad_to_deg = 57.29577951;

    cout << "distance = " << dapos.distance;
    cout << ", angle = " << dapos.angle * Rad_to_deg;
    cout << " degrees\n";
}
```

Notice that the formal variable is type polar. When you pass a polar structure to this function, the structure contents are copied into the dapos structure, and the function then uses that copy in its work. Because dapos is a structure, the function uses the membership (dot) operator (see Chapter 4) to identify structure members.

Next, let's try something more ambitious and write a function that converts rectangular coordinates to polar coordinates. We'll have the function accept a rect structure as its argument and return a polar structure to the calling function. This involves using functions from the math library, so the program has to include the math.h header file. Also, on some systems you have to tell the compiler to load the math library (see Chapter 1, "Getting Started"). You can use the Pythagorean theorem to get the distance from the horizontal and vertical components:

```
distance = sqrt( x * x + y * y)
```

The atan2() function from the math library calculates the angle from the x and y values:

```
angle = atan2(y, x)
```

(There's also an atan() function, but it doesn't distinguish between angles 180 degrees