This is because, as already shown, the implementation of recursion requires some bookkeeping to keep track of the pending recursive calls, and for sufficiently long chains of recursion, the computer simply runs out of memory. This is explained in more detail later in the chapter. This routine also is somewhat more time-consuming than an equivalent loop because the bookkeeping also uses up some time.

Suffice it to say that this particular example does not demonstrate the best use of recursion, since it is so easy to solve the problem without recursion. Most of the good uses of recursion will not exhaust the computer's memory and will be only slightly more time-consuming than nonrecursive implementations. However, recursion will almost always lead to more compact code.

7.3.1 Printing Numbers in Any Base

A good example of how recursion simplifies the coding of routines is number printing. Suppose we want to print out a nonnegative number *N* in decimal, but we do not have a number output method available. However, we can print out one digit at a time. Consider, for instance, how we would print the number `1369`. We would need to print first a `1`, then a `3`, then a `6`, and then a `9`. The problem is that obtaining the first digit is a bit sloppy: Given a number `n`, we need a loop to determine the first digit of `n`. This is in contrast to obtaining the last digit, which is immediately available as `n%10` (which is `n` for `n` less than 10).

Recursion provides a nifty solution. To print out `1369`, we print out `136`, followed by the last digit, `9`. As mentioned, it is easy to print out the last digit using the `%` operator. Printing out all but the number represented by eliminating the last digit is also easy; this is the same problem as printing out `n/10`. Thus it can be done by a recursive call.

The routine in Figure 7.2 implements this printing routine. If `n` is smaller than 10, then line 6 is not executed and only the one digit `n%10` is printed. Otherwise, all but the last digit is printed recursively, and then the last digit is printed.

```
1     // Print n as a Decimal Number
2
3     public static void printDecimal( int n )
4     {
5         if( n >= 10 )
6             printDecimal( n / 10 );
7         printDigit( n % 10 );
8     }
```

Figure 7.2 Recursive routine to print *N* in decimal