```
1     // Print n in any base
2     // Assumes 2 <= base <= 16
3
4     final static String digitTable = "0123456789abcdef";
5
6     public static void printInt( int n, int base )
7     {
8         if( n >= base )
9             printInt( n / base, base );
10        System.out.print( digitTable.charAt( n % base ) );
11    }
```

Figure 7.3 Recursive routine to print *N* in any base

Notice how we have a base case (`n` is a one-digit integer) and that all recursive calls make progress toward the base case because the recursive problem has one less digit. Thus we have satisfied the first two fundamental rules of recursion.

To make our printing routine useful, we extend it to print in any base between 2 and 16.[2] This modification is shown in Figure 7.3. We have introduced a `String` to make the printing of `a` through `f` easier. Each digit is now output by indexing into the `digitTable` array. The `printInt` routine is not robust. If `base` is more than 16, then the index into `digitTable` will be out of the `digitTable` array. If `base` is 0, then an arithmetic error will result when a division by 0 is attempted at line 9.

Failure to make progress means the program does not work.

The most interesting error occurs when `base` is 1. When that happens, the recursive call at line 9 fails to make progress. This is because the two parameters to the recursive call will be identical to the original call. Thus the system will make recursive calls until it eventually runs out of bookkeeping space (and exits less than gracefully with an exception).

A *driver routine* tests the validity of the first call and then calls the recursive routine.

We can make the routine more robust by adding an explicit test for `base`. The problem with that strategy is that the test would be executed during each of the recursive calls to `printInt`, and not just during the first call. Once `base` is valid in the first call, it is silly to retest it, since it does not change in the course of the recursion, and thus must still be valid. One way to avoid this inefficiency is to set up a driver routine. A *driver routine* tests the validity of `base` and then calls the recursive routine. This is shown in Figure 7.4. The use of driver routines for recursive programs is a common technique.

7.3.2 Why It Works

This section shows, somewhat rigorously, that the `printDecimal` algorithm works. Our goal is to verify that the algorithm is correct, so the proof will assume that we have made no syntax errors.

2. Java's `toString` method can take any base, but many languages do not have this built-in capability.