5.29 *(Prime Numbers)* An integer is said to be *prime* if it's divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

a) Write a function that determines whether a number is prime.

b) Use this function in a program that determines and prints all the prime numbers between 2 and 10,000. How many of these numbers do you really have to test before being sure that you've found all the primes?

c) Initially, you might think that $n/2$ is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of n. Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

5.36 *(Recursive Exponentiation)* Write a recursive function `power( base, exponent )` that, when invoked, returns

$$base^{exponent}$$

For example, `power( 3, 4 ) = 3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to 1. *Hint:* The recursion step would use the relationship

$$base^{exponent} = base \cdot base^{exponent - 1}$$

and the terminating condition occurs when `exponent` is equal to `1`, because

$$base^{1} = base$$

5.37 *(Fibonacci Series)* The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms. (a) Write a *nonrecursive* function `fibonacci(n)` that uses type `int` to calculate the *n*th Fibonacci number. (b) Determine the largest `int` Fibonacci number that can be printed on your system. Modify the program of part (a) to use `double` instead of `int` to calculate and return Fibonacci numbers, and use this modified program to repeat part (b).

5.38 *(Towers of Hanoi)* In this chapter, you studied functions that can be easily implemented both recursively and iteratively. In this exercise, we present a problem whose recursive solution demonstrates the elegance of recursion, and whose iterative solution may not be as apparent.

The **Towers of Hanoi** is one of the most famous classic problems every budding computer scientist must grapple with. Legend has it that in a temple in the Far East, priests are attempting to move a stack of golden disks from one diamond peg to another (Fig. 5.34). The initial stack has 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from one peg to another under the constraints that exactly one disk is moved at a time and at no time may a larger disk be placed above a smaller disk. Three pegs are provided, one being used for temporarily holding disks. Supposedly, the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

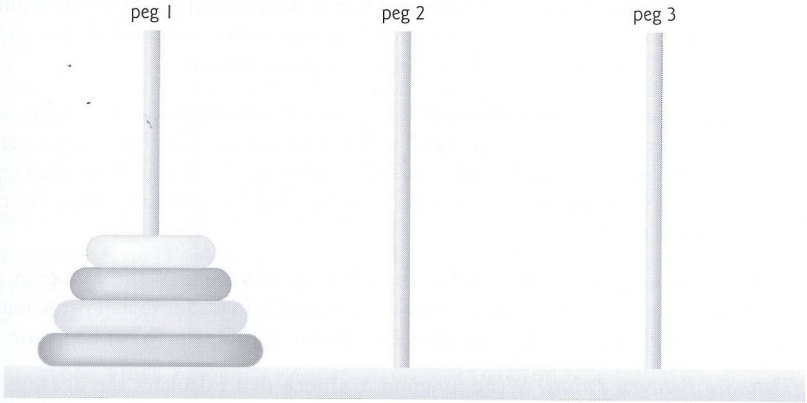

Fig. 5.34 | Towers of Hanoi for the case with four disks.

Let's assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that prints the precise sequence of peg-to-peg disk transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, attacking this problem with recursion in mind allows the steps to be simple. Moving n disks can be viewed in terms of moving only $n - 1$ disks (hence, the recursion), as follows:

a) Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
b) Move the last disk (the largest) from peg 1 to peg 3.
c) Move the $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving $n = 1$ disk (i.e., the base case). This task is accomplished by simply moving the disk, without the need for a temporary holding area. Write a program to solve the Towers of Hanoi problem. Use a recursive function with four parameters:

a) The number of disks to be moved
b) The peg on which these disks are initially threaded
c) The peg to which this stack of disks is to be moved
d) The peg to be used as a temporary holding area

Display the precise instructions for moving the disks from the starting peg to the destination peg. To move a stack of three disks from peg 1 to peg 3, the program displays the following moves:

$1 \rightarrow 3$ (This means move one disk from peg 1 to peg 3.)
$1 \rightarrow 2$
$3 \rightarrow 2$
$1 \rightarrow 3$
$2 \rightarrow 1$
$2 \rightarrow 3$
$1 \rightarrow 3$

5.39 *(Towers of Hanoi: Iterative Version)* Any program that can be implemented recursively can be implemented iteratively, although sometimes with more difficulty and less clarity. Try writing an iterative version of the Towers of Hanoi. If you succeed, compare your iterative version with the recursive version developed in Exercise 5.38. Investigate issues of performance, clarity and your ability to demonstrate the correctness of the programs.