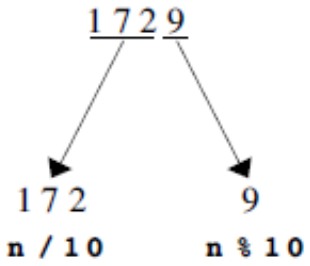
Name:…… Q7

CS2123 Data Structures

(questions are from our textbook)

6. Write a recursive function `DigitSum(n)` that takes a nonnegative integer and returns the sum of its digits. For example, calling `DigitSum(1729)` should return 1 + 7 + 2 + 9, which is 19.

   The recursive implementation of `DigitSum` depends on the fact that it is very easy to break an integer down into two components using division by 10. For example, given the integer 1729, you can divide it into two pieces as follows:



   Each of the resulting integers is strictly smaller than the original and thus represents a simpler case.

7. The **digital root** of an integer *n* is defined as the result of summing the digits repeatedly until only a single digit remains. For example, the digital root of 1729 can be calculated using the following steps:

Step 1:	1 + 7 + 2 + 9	→	19
Step 2:	1 + 9	→	10
Step 3:	1 + 0	→	1

   Because the total at the end of step 3 is the single digit 1, that value is the digital root.

   Write a function `DigitalRoot(n)` that returns the digital root of its argument. Although it is easy to implement `DigitalRoot` using the `DigitSum` function from exercise 6 and a `while` loop, part of the challenge of this problem is to write the function recursively without using any explicit loop constructs.

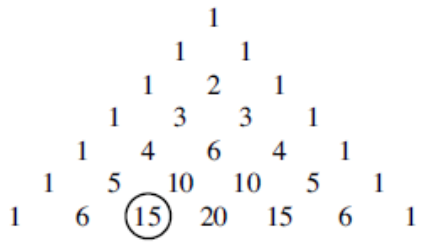
8. The mathematical combinations function $C(n, k)$ is usually defined in terms of factorials, as follows:

$$C(n,k) = \frac{n!}{k! \times (n-k)!}$$

The values of $C(n, k)$ can also be arranged geometrically to form a triangle in which n increases as you move down the triangle and k increases as you move from left to right. The resulting structure,, which is called *Pascal's Triangle* after the French mathematician Blaise Pascal, is arranged like this:

$$\begin{array}{ccccccccc}
 & & & & C(0,0) & & & & \\
 & & & C(1,0) & & C(1,1) & & & \\
 & & C(2,0) & & C(2,1) & & C(2,2) & & \\
 & C(3,0) & & C(3,1) & & C(3,2) & & C(3,3) & \\
C(4,0) & & C(4,1) & & C(4,2) & & C(4,3) & & C(4,4)
\end{array}$$

Pascal's Triangle has the interesting property that every entry is the sum of the two entries above it, except along the left and right edges, where the values are always 1. Consider, for example, the circled entry in the following display of Pascal's Triangle:



This entry, which corresponds to $C(6, 2)$, is the sum of the two entries—5 and 10—that appear above it to either side. Use this relationship between entries in Pascal's Triangle to write a recursive implementation of the **`Combinations`** function that uses no loops, no multiplication, and no calls to **`Fact`**.