The purpose of this exercise is to give you practice writing recursive functions. In addition you
will use command line arguments to enter data in your program.

A recursive function is one that calls itself - this is a very useful technique for some types of
problems. A classic example is the factorial function:

\[ n! = n \times (n - 1) \times \cdots \times 1 \]

The factorial function can be expressed recursively as:

\[ n! = n \times (n - 1)! \]

Recursive mathematical functions are often expressed as recurrence relations. For example, the
recurrence relation for factorial looks like:

\[ n! = \begin{cases} 
1 & \text{if } n = 0 \\
 n \times (n - 1)! & \text{if } n > 0 
\end{cases} \]

A recurrence relation specifies a base (or stopping) case, which can be solved without recursion,
and one or more recursive cases, where the value of the function can be expressed as a smaller or
diminished version of the function itself.

For this lab, you are to write a recursive function that solves the following recurrence relation,
defined for positive \( n \):

\[ f(n) = \begin{cases} 
0 & \text{if } n = 0 \\
f\left(\frac{n}{2}\right) & \text{if } n \text{ is even, } n > 0 \\
1 + f(n - 1) & \text{if } n \text{ is odd, } n > 0 
\end{cases} \]

In addition, you are to obtain the value of \( n \) from the command line. When you type a command,
the shell (or command interpreter) parses the command into a set of strings that can be passed to
the main program. In other words, the main program can have arguments of the form:

\[
\text{int main( int argc, char *argv[]} ...
\]

where \( \text{argc} \) contains the number of command line strings, and \( \text{argv} \) is a pointer to an array con-
taining the (c-style) strings. The count includes the program name itself, and the first string in the
array is the program name.

For example, you might type the following to run your program:

\[$ \ .a.out 10 8 12$\]

In this example, \( \text{argc} \) would have the value 4, the first string (accessed by \( \text{argv}[0] \)) would be "./a.out", the second string would be "10", the third string would be "8", and the last string
would be "12". For this lab, the numeric arguments should be converted to ints (perhaps using
\( \text{atoi} \)) and then passed, one at a time, to your recursive function. Your main program should
output the result from the recursive function for each command line argument.

As usual, submit your code using \( \text{cscheckin} \).