Enter total number of game card choices and number of picks allowed:

49 6
You have one chance in 1.39838e+007 of winning.

Next two numbers (q to quit): 51 6
You have one chance in 1.80095e+007 of winning.

Next two numbers (q to quit): 38 6
You have one chance in 2.76068e+006 of winning.

Next two numbers (q to quit): q
bye

**Program Notes**

The `odds()` function illustrates two kinds of local variables you can have in a function. First, there are the formal parameters (`numbers` and `picks`), which are declared in the function heading before the opening brace. Then come the other local variables (`result`, `n`, and `p`). They are declared in between the braces bounding the function definition. The main difference between the formal parameters and the other local variables is that the formal parameters get their values from the function that calls `odds()`, whereas the other variables get values from within the function.

**Functions and Arrays**

So far the example functions have been simple, using only the basic types for arguments and return values. But functions can be the key to handling more involved types, such as arrays and structures. Let’s take a look now at how arrays and functions get along with each other. Suppose you use an array to keep track of how many cookies each person has eaten at the family picnic. (Each array index corresponds to a person, and the value of the element corresponds to the number of cookies that person ate.) Now you want the total. That’s easy to do; just use a loop to add all the array elements. But adding array elements is such a common task that it makes sense to design a function to do the job. Then, you won’t have to write a new loop every time you have to sum an array.

Let's consider what the function interface involves. Because the function calculates a sum, it should return the answer. If you keep your cookies intact, you can use a function with a type `int` return value. So that the function knows what array to sum, you want to pass the