n **choose** k

In the game of poker a "hand" of five cards is drawn from a "pack" of 52 cards. How many different hands are possible?

This is an example of an important general problem. If we have some number, n, of cards, all different from each other (the pack), and a smaller number, k, drawn from these, in how many ways can this be done?

The answer, well known to mathematicians, is called "n choose k", and is written $\binom{n}{k}$. (In older texts, you may see ${}^nC_k$.) We can derive a formula for it. Imagine putting all the cards down in a line. There are "n factorial" ways of doing this, denoted $n!$ in maths. The value of the factorial of some number x can be calculated using the following pattern.

$$x! = 1 \times 2 \times 3 \times 4 \times \cdots \times (x-1) \times x$$

Now that we have all the cards in a line, we just take the first k of them. Are we done? No, the order in which the k cards of the hand were picked up does not matter in poker. There are $k!$ ways to arrange those cards. The order of the $n-k$ cards we're not interested in doesn't matter either, and there are $(n-k)!$ ways to arrange them.

So the total number of hands is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Task

Write and test a computer program using 64-bit signed integers (`long` in Java, `long long` in C, `Int64` in Swift) that displays the correct value of $\binom{n}{k}$ for given n and k. The program must work for any $0 \le k \le n \le 66$.

The program need not work for $k < 0$ or $k > n$, but if it accepts such values quietly, the right answer is 0.

The values of n and k should be supplied in the command line with output going to standard output. For instance, in Java something like:

```
% java Find_nCk 17 10
19448
```

That is, in response to the UNIX command `java Find_nCk 17 10` your program must write the one number `19448` and a terminating newline and

no other characters whatsoever. The numbers must be written as decimal integers with no commas or other grouping characters.

Your program must not use any multiple precision features of your chosen language such as `BigInteger` in Java. Of course you are not allowed to use Python! Now the largest number representable as a signed 64-bit integer is 9,223,372,036,854,775,807 and the largest result you will be asked for is $\binom{66}{33}$ = 7,219,428,434,016,265,740 but do note that 66! = 544,344,939,077,443,064,003,729,240,247,842,752,644,293,064,388,798,874,532,860,126,869,671,081,148,416,000,000,000,000,000.

(2 points, Pair)

Notes

This is an amusing little task whose main purposes are to (a) force you to notice that machine-sized integers are a pain in the posterior even for quite small problems, (b) give you more practice in testing, and (c) remind you that the way something is *defined* and the way it is *implemented* may be radically different as long as they give the same answers. These days, it probably tests your Web searching skills more than anything else.

There is a cunning way to satisfy the letter of the requirements while violating the spirit. That is to write a program in Lisp, Prolog, Smalltalk, SML, Haskell, or any other language supporting large integers that computes a table and generates a Java program

```
public class Find_nCk {
    private static final long[][] choose = {
        {1L},
        {1L,1L},
        {1L,2L,1L},
        ...
        {1L,66L,2145L,...,2145L,66L,1L}
    };
    public static void main(String[] args) {
        ...
        System.out.println(choose[n][k]);
    }
}
```

That's a quick and easy hack that gets the job done, but it's not what this étude is about. Your program must actually compute the answer.

There's another way to do it that *almost* works. There is a generalisation of the factorial called the gamma function. For integers,

$$\begin{aligned} \Gamma(n) &= (n-1)! \\ n! &= \Gamma(n+1) \end{aligned}$$

The C standard library contains a function lgamma(x) = $\ln\Gamma(x)$, so we could compute

```
(long long)rint(exp(lgamma(n + 1.0) - lgamma(n-k + 1.0)))
```

Why does that only *almost* work? (Hint: how many bits of precision in a signed 64-bit integer? How many in an IEEE double?) If you didn't need exact answers, this would be a good method, but you do, so you mustn't use that method for this étude either.

There are at least two honest ways to solve this problem.

In one of them, the only arithmetic operation you perform on these numbers is addition.

Students have managed to solve this using multiplication and division but the overwhelming majority of students who try that get it wrong.

Testing

Start with small numbers: 0 0; 1 0, 1 1; 2 0, 2 1, 2 2. You will find $\binom{n}{k}$ tabulated in many places on the Web. When you are sure you have the pattern of computation right, test with some inputs that yield large outputs. In fact, since the program is required to work for $0 \le k \le n \le 66$, and that's not a lot, you should try *every* legal input.

While you are not allowed to submit an answer in Python (so we can be sure you are using 64-bit integer arithmetic), nobody said you couldn't write an oracle in Python.