This is a Modular Exponentiation problem. Read the wiki page to learn more - [Wiki](http://en.wikipedia.org/wiki/Modular_exponentiation).

So we have to find the value of (a^b) % c. This can be easily calculated using [modular arithmetic](http://en.wikipedia.org/wiki/Modular_arithmetic) and divide and conquer.

We know the following:

1. ( a \* b ) % c = ( (a%c) \* (b%c) ) %c   
   This came from modular arithmetic.
2. a^b = a \* a^(b-1)
3. If b = x + y, then we can write (a^b) = (a^x) \* (a^y)

Using the observations above, we can write a divide and conquer algorithm.

long long bigmod ( long long a, long long b, long long c ) {

if ( b == 0 ) return 1;

if ( b % 2 == 0 ) {

long long x = bigmod ( a, b / 2, c );

x = ( x \* x ) % c;

return x;

}

else {

return ( a \* bigmod ( a, b - 1, c ) ) % c;

}

}

This is a classical algorithm for finding bigmod. But notice that this is still not enough to solve the problem.

x = ( x \* x ) % c;

Here, since c can be as large as 10^18, the value of x can be (1018 - 1 ). When we multiply x with x, it will overflow. For example, try with input a = 1018 - 1, b = 2, c = 1018.

So how do we calculate ( x \* x ) % c, without overflowing? Again, we use the same divide and conquer approach. Only this time we add instead of multiplying. Lets call it bigmul() function.

long long bigmul ( long long a, long long b, long long c ) {

if ( b == 0 ) return 1;

if ( b % 2 == 0 ) {

long long x = bigmul ( a, b / 2, c );

x = ( x + x ) % c;

return x;

}

else {

return ( a + bigmul ( a, b - 1, c ) ) % c;

}

}

So our final algorihtm will be:

long long finalBigmod ( long long a, long long b, long long c ) {

if ( b == 0 ) return 1;

if ( b % 2 == 0 ) {

long long x = finalBigmod ( a, b / 2, c );

x = bigmul ( x, x, c );

return x;

}

else {

return bigmul ( a, finalBigmod ( a, b - 1, c ), c );

}

}