**Modulo 10^9+7 (1000000007)**

In most programming competitions, we are required to answer the result in 10^9+7 modulo.

The reason behind this is, if problem constraints are large integers, only efficient algorithms can solve them in an allowed limited time.

**What is modulo operation:** 

The remainder obtained after the division operation on two operands is known as modulo operation. The operator for doing modulus operation is **‘%’**.

For ex: a % b = c which means, when a is divided by b it gives the remainder c, 7%2 = 1, 17%3 = 2.

**Why do we need modulo:** 

* The reason of taking Mod is to prevent integer overflows.
* The largest integer data type in C/C++ is *unsigned long long int* which is of 64 bit and can handle integer from 0 to (2^64 – 1).
* But in some problems where the growth rate of output is very high, this high range of unsigned long long may be insufficient.
* Suppose in a 64 bit variable ‘A’, 2^62 is stored and in another 64 bit variable ‘B’, 2^63 is stored. When we multiply A and B, the system does not give a runtime error or exception.
* It just does some junk computation and stores the junk result because the bit size of the result comes after multiplication overflows.
* In some of the problems, to compute the result modulo inverse is needed and this number helps a lot because it is prime. Also, this number should be large enough otherwise modular inverse techniques may fail in some situations.

Due to these reasons, problem setters require to give the answer as a result of modulo of some number **N**.   
There are certain criteria on which the value of N depends: 

1. It should just be large enough to fit in the largest integer data type i.e it makes sure that there is no overflow in the result.
2. It should be a prime number because if we take mod of a number by Prime the result is generally spaced i.e. the results are very different results in comparison to mod the number by non-prime, that is why primes are generally used for mod.

10^9+7 fulfills both the criteria. It is the first 10-digit prime number and fits in int data type as well.

In fact, any prime number less than 2^30 will be fine in order to prevent possible overflows.

**How modulo is used:**   
A few distributive properties of modulo are as follows: 

1. ( a + b) % c = ( ( a % c ) + ( b % c ) ) % c
2. ( a \* b) % c = ( ( a % c ) \* ( b % c ) ) % c
3. ( a – b) % c = ( ( a % c ) – ( b % c ) ) % c

So, modulo is distributive over +, \* and – but not over /

NOTE: The result of ( a % b ) will always be less than b.

In the case of computer programs, due to the size of variable limitations, we perform modulo M at each intermediate stage so that range overflow never occurs.

Example:

a = 145785635595363569532135132

b = 3151635135413512165131321321

c = 999874455222222200651351351

m = 1000000007

Print (a\*b\*c)%m.

Method 1:

First, multiply all the number and then take modulo:

(a\*b\*c)%m = (459405448184212290893339835148809515332440033400818566717735644307024625348601572) %

1000000007

a\*b\*c does not fit even in the unsigned long long int

due to which system drop some of its most significant digits. Therefore, it gives the wrong answer.

(a\*b\*c)%m = 798848767

**Method 2:**

Take modulo at each intermediate steps:

i = 1

i = (i\*a) % m // i = 508086243

i = (i\*b) % m // i = 144702857

i = (i\*c) % m // i = 798848767

i = 798848767

Method 2 always gives the correct answer.

Also, Just to bring to your notice, an add on, the modulo works only with int data type and

not with float/double. So, never use pow(10,9) + 7 for writing 10^9 + 7.

since, power returns a double.

In competitive programming problems, sometimes they want answer in the form of (ans)%(10^9+7),

this means that the actual answer to a problem lies above the range of 64-bit integer

which is not possible to calculate so they want only the remainder left behind your actual answer.

Therefore to tackle with this problem, you just simply learn two basic rules:

1.) (a+b)%m = (a%m+b%m)%m.

2.) (a\*b)%m = (a%m\*b%m)%m.

so next time when you comes to this situation just apply the rules given above at each step of

your calculation.

for ex: to calculate the factorial of a large number N (1<=N<=10^9) modulus (10^9+7)

so the solution is :

long M = 1000000007;

long fact = 1;

for (long i = 1; i <= N; i++)

fact = (fact\*i) % M; // Now f can never exceed 10^9+7

print(fact);

**Refer to FactTest.java**