as you are in class 7 you already know some mathematics. so i am going to give some contest related instructions that matches your math syllabus.

**GCD / LCM**

gcd means greatest common divisor.
lcm means least common multiple.

gcd(x, y) = gcd of x and y
lcm(x, y) = lcm of x and y

in c programming we shall use this 2 functions

```
long long gcd(long long x, long long y)
long long lcm(long long x, long long y)
```

they take 2 parameters and returns gcd or lcm in long long form.

**relation between gcd and lcm**

gcd(x, y) * lcm(x, y) = x * y

using this formula we can find the lcm of 2 numbers if their gcd is known.

**property of gcd and lcm**

gcd(a, b, c) = gcd( gcd(a, b), c ) = gcd( a, gcd(b,c) )

lcm(a, b, c) = lcm( lcm(a, b), c ) = lcm( a, lcm(b,c) )

*see gcd_lcm.c to learn about them inside codes folder.

**minimizing(kata-kati) a fraction**

if you have to consider a fraction(vog-nangsho) in C then you can take one variable for numerator(lob) and another for denominator(hor).

say it is 3/12

so you can code like,

```
int a = 3, b = 12;
```

you can also take an array of size 2 and consider the $0^{th}$ index as numerator and $1^{st}$ index as denominator.

like,

```
long long f[2];
f[0] = 3;   f[1] = 12;
```

to minimize a fraction you have to divide both numerator and denominator by their gcd.

so to minimize first find,
now,
done !

```
long long k = gcd(f[0], f[1]);
f[0] = f[0]/k;    f[1] = f[1]/k;
```

**prime numbers**

prime number is known to you. many difficult problems exist in contest arena related to primes. but i hope if you face any it will be simpler.

the first question anyone can ask about prime is – is this number a prime ? yes or no. if you can answer, then the later part of the problem becomes easy.

so how to decide if a number is prime or not ?

**method 1**

say, a value n is given we have to test its primality. we can easily start from 2 to n-1 and test if n can be divided by any of them. if any of them can divide n we can stop the test there and report n is composite. if none of them can divide n we can report n is prime.

this process is done in prime_method_1.c file.

see the code and try to understand the isPrime(long long n) function.

**method 2**

method 1 is very slow and time consuming. say the number n = 100. it can be divided by 2 so its not prime.

if n = 113, it is odd.
it will never be divisible by any even numbers like, 4, 6, 8 etc. why ? because if it is divisible by any even number, then it is already divisible by 2 !

so we don't need to test with any even value. this method is made clear in prime_method_2.c file. look at that. we are jumping with odd numbers.

**method 3**

from number theory you know that, to know if n is prime or not, it is enough to check its divisibility from 2 to its square root.

e.g. n = 145,  sqrt(n) = sqrt(145) = 12.0459

so it is enough to test its divisibility from 2 to 12.

this method is made clear in prime_method_3.c.

see there is a slight difference inside the for loop of isPrime() function.

this mehtod is faster and easiest for you.

**method 4**

method 3 is faster but not so fast. there is a method called Sieve of Eratosthenes which is mostly used in contest. the main idea is to generate all the prime from 0 to a higher range at the beginning of the program in an array. then using the array we can directly tell about the primality of a number.

why is this useful ?

when you are using method 3 or previous ones, there for each number you are starting the test from 2 until the number or its square root.

e.g. n = 100
you run the test counting from 2 till 99 (method 1 or 2) or till square root of 99.

again if n == 50 you do the same from 2 till 49 or sqrt(50)

the counting process goes again and again.

but in sieve of eratosthenes we do the counting only once.

this method is somewhat complex. i will explain in brief how it works.

in school you did it.

first write from 1 to the range of number you want to find primes within. say it is 20.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

now cross 1 as it is not prime.

x 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

now you get 2 at the start. so it is a prime. now cross all that are divisible by 2. it becomes,

x 2 3 x 5 x 7 x 9 x 11 x 13 x 15 x 17 x 19 x

now you get another prime 3.

now cross all that are divisible by 3 if not crossed already. 9 and 15 are crossed out. it becomes,

x 2 3 x 5 x 7 x x x 11 x 13 x x x 17 x 19 x

then 5 is another prime found. cross any number divisible by 5. none found. because, 10, 15 and 20 are already crossed. so 7 is prime. none is found as a victim of 7.
likewise, 11, 13, 17 and 19 are found as primes.

the algorithm given is prime_method_4.c works in this way and also uses the square root technique described in method 3.

**how to use this method ?**

take a global array called P.

then immediately after main run the function named sieve().

done. the method will generate a list of prime from 0 to 1000000

now, the thing happened here is,

if n is a prime number then p[n] == 1
if n is not prime then P[n] == 0

you can use an if condition to test it. see prime_method_4.c to understand.

**in contest try to use this method 4. it must work within the time limit. but i hope method 3 will work for your contest. if method 3 fails and gives TLE you must use method 4 – the sieve().**

***i declared the size of P array 1000000 + 10  for safety. its always good to keep some extra space from the maximum limit. it helps us to ignore mistakes.**

**Fibonacci series**

fibonacci series is

1 1 2 3 5 8 13 21 ...

every element is the sum of previous two element and by default the first two element is 1.

a program named fibo.c is given to print first n numbers of the fibonacci series. it may help.