VIII MARATON INTERNA DE PROGRAMACION

UNIVERSIDAD KONRAD LORENZ

22 de Mayo de 2010

PROBLEMAS

Elaborado por: Hector Florez
Basado de www.acis.org.co, www.acm.org

Problem 1

Middle Primes

Source file name: mprime.c, mprime.cpp or mprime.java
Input: mprime.in
Output: standar output

A prime number is a counting number (1, 2, 3, ...) that is evenly divisible only by 1 and itself. In this problem you are to write a program that will cut some number of prime numbers from the list of prime numbers between (and including) 1 and N.

Your program will read in a number N, determine the list of prime numbers between 1 and N, and print the C*2 prime numbers from the center of the list if there are an even number of prime numbers or (C*2)-1 prime numbers from the center of the list if there are an odd number of prime numbers in the list.

Input

Each input set will be on a line by itself and will consist of 2 numbers. The first number ($1 \le N \le 1000$) is the maximum number in the complete list of prime numbers between 1 and N. The second number ($1 \le C \le N$) defines the C*2 prime numbers to be printed from the center of the list if the length of the list is even; or the (C*2)-1 numbers to be printed from the center of the list if the length of the list is odd.

Output

For each input set, you should print the number N beginning in column 1 followed by a space, then by the number C, then by a colon (:), and then by the center numbers from the list of prime numbers as defined above. If the size of the center list exceeds the limits of the list of prime numbers between 1 and N, the list of prime numbers between 1 and N (inclusive) should be printed. Each number from the center of the list should be preceded by exactly one blank. Each line of output should be followed by a blank line.

Sample input	**Sample Output**
21 2 18 2 18 18 100 7	21 2: 5 7 11 18 2: 3 5 7 11 18 18: 1 2 3 5 7 11 13 17 100 7: 13 17 19 23 29 31 37 41 43 47 53 59 61 67

Problem 2

Emoticons :-)

Source file name: emoticons.c, emoticons.cpp or emoticons.java
Input: emoticons.in
Output: standar output

Emoticons are used in chat and e-mail conversations to try to express the emotions that printed words cannot. This may seem like a nice feature for many, but a lot of people find it really annoying and wants to get rid of emoticons.

George is one of those people. He hates emoticons so bad, that he is preparing a plan to remove all emoticons from all e-mails in the world. Since you share his visionary plans, you are preparing a special program to help him.

Your program will receive the list of emoticons to proscribe. Each emoticon will be a string of characters not including any whitespace. You will also receive several lines of text. What you need to do is change some characters of the text into spaces to ensure no emoticon is left on the text. For an emoticon to be considered to appear in the text it has to appear in a single line and be made of consecutive characters.

To help George's plan remain secret as long as possible, you need to do your job with the minimum possible amount of character changes.

Input

The input file contains several test cases. Each test case consists of several lines. The first line of each test case will contain two integers separated by a single space: N, the number of emoticons to proscribe, and M, the number of lines the text has. The next N lines contain one emoticon each, a non-empty string of at most 15 characters. Each of the last M lines of the test case contains a line of text of at most 80 characters. You can assume 1 <= N, M <= 100. Valid input characters for emoticons are uppercase and lowercase letters, digits and the symbols "!?.,:;-_'#$%&/=*+(){}[]" (quotes for clarity).

Each line of the text may contain the same characters with the addition of the space character. The input is terminated by N = M = 0. The input must be read from standard input.

Output

For each test case, output exactly one line containing a single integer that indicates the minimum number of changes you need to make to the entire text to ensure no emoticon on the list appears in it.

Sample input	Sample Output
4 6 :-) :-((-:)-: Hello uncle John! :-) :-D I am sad or happy? (-:-(? I feel so happy, my head spins (-:-)(-:-)(-:-)(-:-) :-) (-: :-) but then sadness comes :-(Loves you, Joanna :-))))) 3 1 :)):)) :):)):)):)):(:((:(((:):) 0 0	11 8

Problem 3

3n + 1

Source file name: three.c, three.cpp or three.java
Input: three.in
Output: standar output

Consider the following algorithm to generate a sequence of numbers. Start with an integer n. If n is even, divide by 2. If n is odd, multiply by 3 and add 1. Repeat this process with the new value of n, terminating when n = 1. For example, the following sequence of numbers will be generated for n = 22:

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured (but not yet proven) that this algorithm will terminate at n = 1 for every integer n. Still, the conjecture holds for all integers up to at least 1,000,000.

For an input n, the cycle-length of n is the number of numbers generated up to and including the 1. In the example above, the cycle length of 22 is 16. Given any two numbers i and j, you are to determine the maximum cycle length over all numbers between i and j, including both endpoints.

Input

The input will consist of a series of pairs of integers i and j, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

Output

For each pair of input integers i and j, output i, j in the same order in which they appeared in the input and then the maximum cycle length for integers between and including i and j. These three numbers should be separated by one space, with all three numbers on one line and with one line of output for each line of input.

Sample input	Sample Output
1 10 100 200 201 210	1 10 20 100 200 125 201 210 89

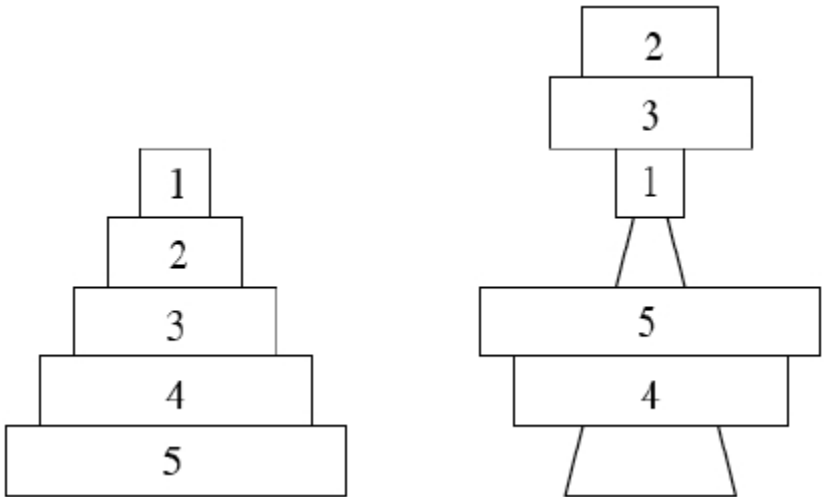
Problem 4

Stacking Tower

Source file name: tower.c, tower.cpp or tower.java
Input: tower.in
Output: standar output

One of the most common children's toys is a stacking tower, which consists of a series of rings of different sizes and a tapered rod which can hold the rings. The rings and rod are designed so that when the rings are placed in descending order by size, they fit exactly on the rod. Further, each ring, if placed by itself on the rod will go no lower than its position when all rings are on the rod.

The diagram on the left shows an example of this toy which uses five rings, numbered 1 (smallest) to 5 (largest)



Unless endowed with super{genius mental faculties, most infants will place the rings in a random order on the rod, which results in some rings sticking over the top of the rod. The above diagram on the right shows the result of one such random placement. Your job will be to determine the number of rings which sit above the rod given a random ordering of the rings. You may assume that the rings may be stacked arbitrarily high without falling over.

Input

Input will consist of multiple problem instances. Each instance consists of a single line of the form n r_1 r_2 … r_n. The positive integer n specifies the number of rings (<= 100), and the remaining n integers give the order the rings are put on the rod. A value of n = 0 will terminate input.

Output

For each problem instance, you should output one line containing an integer indicating the number of rings sticking over the top of the rod.

Sample input	Sample Output
5 5 4 3 2 1 5 4 5 1 3 2 8 1 2 3 4 5 6 7 8 20 11 10 19 7 12 14 5 2 3 1 8 6 13 17 18 9 4 20 15 16 0	0 2 7 11

Problem 5

Factorial

Source file name: factorial.c, factorial.cpp or factorial.java
Input: factorial.in
Output: standar output

Input

The input contains several test cases. Each line contains one case. Each case contain one number n that 1<=n<=15

Output

For each case, calculate the factorial

Sample input	Sample Output
5 6	120 720

Problem 6

Quicksum

Source file name: quicksum.c, quicksum.cpp or quicksum.java
Input: quicksum.in
Output: standar output

A checksum is an algorithm that scans a packet of data and returns a single number. The idea is that if the packet is changed, the checksum will also change, so checksums are often used for detecting transmission errors, validating document contents, and in many other situations where it is necessary to detect undesirable changes in data. For this problem, you will implement a checksum algorithm called Quicksum. A Quicksum packet allows only uppercase letters and spaces. It always begins and ends with an uppercase letter. Otherwise, spaces and letters can occur in any combination, including consecutive spaces.

A Quicksum is the sum of the products of each character's position in the packet times the character's value. A space has a value of zero, while letters have a value equal to their position in the alphabet. So, A = 1, B = 2, etc., through Z = 26. Here are example Quicksum calculations for the packets "ACM" and "MID CENTRAL":

ACM: $1 \times 1 + 2 \times 3 + 3 \times 13 = 46$
MID CENTRAL: $1 \times 13 + 2 \times 9 + 3 \times 4 + 4 \times 0 + 5 \times 3 + 6 \times 5 + 7 \times 14 + 8 \times 20 + 9 \times 18 + 10 \times 1 + 11 \times 12 = 650$

Input

The input consists of one or more packets followed by a line containing only # that signals the end of the input. Each packet is on a line by itself, does not begin or end with a space, and contains from 1 to 255 characters.

Output

For each packet, output its Quicksum on a separate line in the output.

Sample input	Sample Output
ACM MID CENTRAL REGIONAL PROGRAMMING CONTEST ACN A C M ABC BBC #	46 650 4690 49 75 14 15