PLU February 2013 Programming Contest

Advanced Problems

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 10.

2. Problems will have either no input or will read input from standard input (stdin, cin, System.in -- the keyboard). All output should be to standard output (the monitor).

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

Number	**Name**
Problem 1	Call For Help
Problem 2	LCM
Problem 3	Parity Bit
Problem 4	Good Versus Evil
Problem 5	Factoring
Problem 6	Seating Chart
Problem 7	Duplicate SSN
Problem 8	Transport
Problem 9	Coin Row
Problem 10	Coin Collection
Problem 11	Open Interval

Good luck!

1. Call For Help

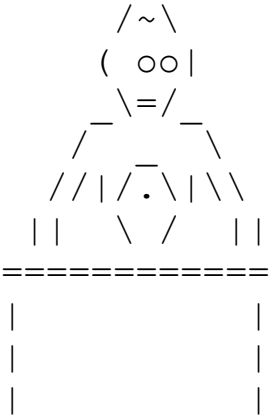
Problem Description: C3PO is in trouble. Your only chance to save him is to broadcast this ASCII picture depicting his current predicament.

Input: none

Output: ASCII picture exactly has shown below. The last 3 rows have 10 spaces between the two '|' characters.

Sample Input: none

Sample Output:



2. LCM

Problem Description: A fundamental step in adding and subtracting fractions is finding the least common multiple. Doing this by hand is difficult for some children (and some adults). You will write a program that finds the least common multiple of two positive integers. The least common multiple of two integers **a** and **b** is the smallest positive integer that is divisible by both **a** and **b**.

Input: The first line of input will be a positive integer, n, indicating the number of problem sets. Each problem set consist of two positive integers, separated by one or more spaces. There are no blank lines in the input.

Output: For each problem set print the least common multiple of the two positive integers. Print each least common multiple on a separate line.

Sample Input:

```
3
15 21
33 22
9 10
```

Sample Output:

```
105
66
90
```

3. Parity Bit

Problem Description: A parity bit, or check bit, is a bit that is added to the end of a sequence of bits for error detection. In telecommunications and computing all data is transformed into a sequence of zeros and ones. For this problem you may assume that all information is coded using 8 bits, with the first 7 bits used for data and the last bit used as the parity bit. The parity bit is set to one if the number of ones in the preceding 7 bits is odd, and the parity bit is set to zero if the number of ones in the preceding 7 bits is even.

7 bits of data (count of 1 bits)		8 bits including parity bit
0000000	(0)	00000000
0101001	(3)	01010011
0111001	(4)	01110010
0110111	(5)	01101111
1001100	(2)	10011000

The parity bit can be used to determine if the data was transmitted incorrectly. If the parity bit is incorrect (does not match the parity of the previous 7 bits), then a parity error occurred. Your job is to read lines of data and determine the number of parity errors on each line.

Input: The first line will be a positive integer indicating the number of lines of data transmitted. Each line of data will be a sequence of zeros and ones and the length of the line will be a multiple of 8 and no longer than 64.

Output: For each data transmission line print the number of parity errors found in the data transmission.

Sample Input:

```
3
00000000
01010010
0000000101010010
```

Sample Output:

```
0
1
2
```

4. Good Versus Evil

Problem Description: Middle Earth is about to go to war. The forces of good will have many battles with the forces of evil. Different races will certainly be involved. Each race has a certain `worth' when battling against others. On the side of good and evil we have the following races, with their associated worth.

On The Side of Good	On The Side of Evil
Hobbits - 1	Orcs - 1
Men - 2	Men - 2
Elves - 3	Wargs - 2
Dwarves - 3	Goblins - 2
Eagles - 4	Uruk Hai - 3
Wizards – 10	Trolls - 5
	Wizards - 11

Although weather, location, supplies and valor play a part in any battle, if you add up the worth of the side of good and compare it with the worth of the side of evil, the side with the larger worth will tend to win. Thus, given the count of each of the races on the side of good, followed by the count of each of the races on the side of evil, determine which side wins.

Input The first line of input will contain an integer greater than 0 signifying the number of battles to process. Information for each battle will consist of two lines of data as follows. First, there will be a line containing the count of each race on the side of good. Each entry will be separated by a single space. The values will be ordered as follows: Hobbits, Men, Elves, Dwarves, Eagles, Wizards.
The next line will contain the count of each race on the side of evil in the following order: Orcs, Men, Wargs, Goblins, Uruk Hai, Trolls, Wizards. All values are non-negative integers. The resulting sum of the worth for each side will not exceed the limit of a 32-bit integer.

Output: For each battle, print “Battle" followed by a single space, followed by the battle number starting at 1, followed by a “:", followed by a single space. Then print “Good triumphs over Evil" if good wins. Print “Evil eradicates all trace of Good" if evil wins. If there is a tie, then print “No victor on this battle field".

Sample Input:

```
3
1 1 1 1 1 1
1 1 1 1 1 1 1
0 0 0 0 0 10
0 1 1 1 1 0 0
1 0 0 0 0 0
1 0 0 0 0 0 0
```

Sample Output:

```
Battle 1: Evil eradicates all trace of Good
Battle 2: Good triumphs over Evil
Battle 3: No victor on this battle field
```

5. Factoring

Problem Description: A prime number (or a prime) is an integer greater than 1 that has no positive divisors other than 1 and itself. An integer greater than one that is not prime is called composite. The fundamental theorem of arithmetic, also called the unique factorization theorem or the unique-prime-factorization theorem, states that every integer greater than 1 is either prime or is the product of prime numbers, and that, although the order of the primes in the second case is arbitrary, the primes themselves are not. If we place the prime factors of a composite number in ascending order, then we have a unique prime factorization. Given an integer greater than one either state that the integer is prime or give the unique prime factorization of the composite number.

Input: The first line will be a positive integer, n, indicating the number of input lines that follow. Each input line will contain one integer greater than 1 and less than 2^{31}.

Output: For each line of input, other than the first, print the number followed by a colon and either the statement that the number is prime or the unique prime factorization of the composite number. Primes in the unique factorization must be separated by a single blank space, and there must be one blank space after the colon.

Sample Input:

```
3
55
7877
3366
```

Sample Output:

```
55: 5 11
7877: prime
3366: 2 3 3 11 17
```

6. Seating Chart

Problem Description: Bilbo's birthday is coming up, and Frodo and Sam are in charge of all the party planning! They have invited all the hobbits of Middle Earth to the party, and everyone will be sitting in a single row at an extremely long dining table.
However, due to poor communication, Frodo and Sam have each independently put together a seating chart for all the hobbits at the dining table. Help Frodo and Sam find out how similar their seating charts are by counting the total number of distinct pairs of hobbits who appear in different orders in the two charts.

Input: The input will contain multiple test cases. Each test case begins with a single line containing an integer N ($1 \leq N \leq 100{,}000$) indicating the number of hobbits. The next two lines represent Frodo's and Sam's seating charts, respectively. Each seating chart is specified as a single line of N unique alphabetical strings; the set of strings in each line are guaranteed to be identical. The end-of-input is denoted by a line containing the number 0.

Output: For each input test case, output a single integer denoting, out of the (N * (N-1)) / 2 distinct pairs of hobbits, how many pairs appear in different orders in Frodo's and Sam's seating arrangements.

Sample Input:

```
3
Frodo Sam Bilbo
Sam Frodo Bilbo
5
A B C D E
B A D E C
0
```

Sample Output:

```
1
3
```

7. Duplicate SSN

Problem Description: The U.S. Social Security Administration has made a terrible mistake. When assigning new Social Security numbers (SSN) to U.S. citizens they accidently assigned some duplicate numbers. Fortunately, they caught this mistake early and have a list all the possible duplicates. The Social Security Administration must contact all the people that do not have a unique SSN. Your job is to read the list of Social Security numbers and find all the duplicates. You will then print the list of duplicates in ascending order.

Input: The input is a list of Social Security numbers, one per line. Input will be terminated with the Social Security number 000-00-0000, which is not a duplicate.

Output: A list of all Social Security numbers, in ascending order, that appear more than once as input.

Sample Input:

```
555-44-6666
111-99-4444
012-00-1111
888-98-9086
555-44-6666
234-54-3425
012-00-1111
555-44-6666
000-00-0000
```

Sample Output:

```
012-00-1111
555-44-6666
```

8. Transport

Problem Description: You have a transport plane that has to deliver items to a remote location. You would like to load all the items on the plane, but you can not exceed the plane's weight capacity, W. Given n items of known weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$ find the most valuable subset of the items that fit into the plane without exceeding the capacity, W.

Input: The first line of input will be a positive integer indicating the number of problem sets. Each problem set will start with a line that has two positive integers, n and W, where n is the number of items and W is the capacity of the plane. The next n lines will have two integers, w and v, where w is the weight and v is the value of the item. All weights and values will be positive integers, and the number of items in any problem set will be no more than 20.

Output: For each problem set print, on a separate line, the total value of the most valuable subset that the plane can transport without exceeding its capacity W.

Sample Input:

```
2
3 5
2 3
2 2
3 3
4 10
7 42
3 12
4 40
5 25
```

Sample Output:

```
6
65
```

9. Coin Row

Problem Description: There is a row of n coins whose values are some positive integers, c_1, c_2, …, c_n, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two adjacent coins can be selected.

Input: The first line of input will be a positive integer, **n**, indicating the number of coin rows that follow. Following the positive integer will be **n** coin rows, one per line. Each coin row will be a list of positive integers separated by one or more blanks, and there will be at most 20 coins in any one row.

Output: For each coin row print, on a separate line, the maximum amount of money you can pick up subject to the constraint that no two adjacent coins are selected. The output should be an unformatted positive integer.

Sample Input:

```
2
5 1 2 10 6 2
22 55 66 55 15 10
```

Sample Output:

```
17
120
```

10. Coin Collection

Problem Description: Coins are placed on the squares of an **n**x**m** board, with no more than one coin per square. A robot, located in the upper left square of the board, needs to collect as many coins as possible and bring them to the bottom right square. On each step the robot can move either one cell to the right or one cell down from its current location. When the robot visits a cell with a coin, it always picks up the coin. Design an algorithm to find the maximum number of coins the robot can collect.

For example, given the 5x7 board below the robot can collect at most 6 coins.

C					**C**	**C**
	C	**C**	**C**			
						C
C		**C**	**C**			
	C				**C**	

Input: The first line of input will be a positive integer indicating the number of problem sets. Each problem set will start with a line that has two positive integers, $n \leq 50$ and $m \leq 50$, which represent the board size. The next n lines will contain m characters that are either an X or C. The X will represent a blank and the C a coin. Thus, the example 5x7 board above is represented by problem set

```
5 7
CXXXXCC
XCCCXXX
XXXXXXC
CXCCXXX
XCXXXCX
```

Output: For each problem set print, on a separate line, the maximum number of coins the robot can collect.

Sample Input:

```
2
5 7
CXXXXCC
XCCCXXX
XXXXXXC
CXCCXXX
XCXXXCX
4 4
XXXX
CCCC
XXXX
CCCC
```

Sample Output:

```
6
5
```

11. Open Intervals

Problem Description: Given n open intervals (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) on the number line, each representing start and end times of some activity requiring the same resource, the goal is to find the largest number of these intervals so that no two of them overlap.

Input: The input will contain multiple test cases. Each test case starts with a single line containing a positive integer, $n \leq 50$, indicating the number of intervals. The next n lines each contain a pair of positive integers separated by one or more blanks. Each pair of integers represents one interval. The end-of-input is denoted by a line containing 0.

Output: For each test case print the largest number of intervals that do not overlap.

Sample Input:

```
5
10 12
2 6
5 8
3 9
1 4
2
1 3
3 5
0
```

Sample Output:

```
3
2
```