10104 - Euclid Problem

Time limit: 3.000 seconds

The Problem

From Euclid it is known that for any positive integers A and B there exist such integers X and Y that $AX+BY=D$, where D is the greatest common divisor of A and B. The problem is to find for given A and B corresponding X, Y and D.

The Input

The input will consist of a set of lines with the integer numbers A and B, separated with space ($A,B<1000000001$).

The Output

For each input line the output line should consist of three integers X, Y and D, separated with space. If there are several such X and Y, you should output that pair for which $|X|+|Y|$ is the minimal (primarily) and $X<=Y$ (secondarily).

Sample Input

```
4 6
17 17
```

Sample Output

```
-1 1 2
0 1 17
```

10235 - Simply Emirp

Time limit: 3.000 seconds

An integer greater than 1 is called a prime number if its only positive divisors (factors) are 1 and itself. Prime numbers have been studied over the years by a lot of mathematicians. Applications of prime numbers arise in Cryptography and Coding Theory among others.

Have you tried reversing a prime ? For most primes, you get a composite (43 becomes 34). *An Emirp (Prime spelt backwards) is a Prime that gives you a different Prime when its digits are reversed*. For example, 17 is *Emirp* because 17 as well as 71 are Prime. In this problem, you have to decide whether a number *N* is Non-prime or Prime or *Emirp*. Assume that *1<N<1000000*.

Interestingly, Emirps are not new to NTU students. We have been boarding 199 and 179 buses for quite a long time!

Input

Input consists of several lines specifying values for *N*.

Output

For each *N* given in the input, output should contain one of the following:

1. "***N*** `is not prime.`", if *N* is not a Prime number.
2. "***N*** `is prime.`", if *N* is Prime and *N* is not *Emirp*.
3. "***N*** `is emirp.`", if *N* is *Emirp*.

Sample Input

```
17
18
19
179
199
```

Sample Output

```
17 is emirp.
18 is not prime.
19 is prime.
179 is emirp.
199 is emirp.
```

10139 - Factovisors

Time limit: 3.000 seconds
The factorial function, n! is defined thus for n a non-negative integer:

```
0! = 1
n! = n * (n-1)!   (n > 0)
```

We say that a divides b if there exists an integer k such that

```
k*a = b
```

The input to your program consists of several lines, each containing two non-negative integers, n and m, both less than 2^31. For each input line, output a line stating whether or not m divides n!, in the format shown below.

Sample Input

```
6 9
6 27
20 10000
20 100000
1000 1009
```

Output for Sample Input

```
9 divides 6!
27 does not divide 6!
10000 divides 20!
100000 does not divide 20!
1009 does not divide 1000!
```

10622 - Perfect P-th Powers

Time limit: 3.000 seconds
We say that *x* is a perfect square if, for some integer *b*, *x* = *b*^2. Similarly, *x* is a perfect cube if, for some integer *b*, *x* = *b*^3. More generally, *x* is a perfect p*th* power if, for some integer *b*, *x* = *b*^*p*. Given an integer *x* you are to determine the largest *p* such that *x* is a perfect p*th* power.

Each test case is given by a line of input containing *x*. The value of *x* will have magnitude at least 2 and be within the range of a (32-bit) *int* in C, C++, and Java. A line containing 0 follows the last test case.

For each test case, output a line giving the largest integer *p* such that *x* is a perfect p*th* power.

Sample Input

```
17
1073741824
25
0
```

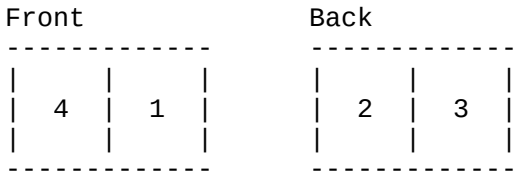
Output for Sample Input

```
1
30
2
```

637 - Booklet Printing

When printing out a document, normally the first page is printed first, then the second, then the third, and so on until the end. However, when creating a fold-over booklet, the order of printing must be altered. A fold-over booklet has four pages per sheet, with two on the front and two on the back. When you stack all the sheets in order, then fold the booklet in half, the pages appear in the correct order as in a regular book.

For example, a 4-page booklet would print on 1 sheet of paper: the front will contain page 4 then page 1, and the back will contain page 2 then page 3.



Your task is to write a program that takes as input the number of pages to be printed, then generates the printing order.

Input
The input file contains one or more test cases, followed by a line containing the number 0 that indicates the end of the file.

Each test case consists of a positive integer *n* on a line by itself, where *n* is the number of pages to be printed; *n* will not exceed 100.

Output
For each test case, output a report indicating which pages should be printed on each sheet, exactly as shown in the example. If the desired number of pages does not completely fill up a sheet, then print the word `Blank` in place of a number. If the front or back of a sheet is entirely blank, do not generate output for that side of the sheet.

Output must be in ascending order by sheet, front first, then back.

Sample Input
```
1
14
4
0
```
Sample Output
```
Printing order for 1 pages:
Sheet 1, front: Blank, 1
Printing order for 14 pages:
Sheet 1, front: Blank, 1
Sheet 1, back : 2, Blank
Sheet 2, front: 14, 3
Sheet 2, back : 4, 13
Sheet 3, front: 12, 5
Sheet 3, back : 6, 11
Sheet 4, front: 10, 7
Sheet 4, back : 8, 9
Printing order for 4 pages:
Sheet 1, front: 4, 1
Sheet 1, back : 2, 3
```

138 - Street Numbers

Time limit: 3.000 seconds

A computer programmer lives in a street with houses numbered consecutively (from 1) down one side of the street. Every evening she walks her dog by leaving her house and randomly turning left or right and walking to the end of the street and back. One night she adds up the street numbers of the houses she passes (excluding her own). The next time she walks the other way she repeats this and finds, to her astonishment, that the two sums are the same. Although this is determined in part by her house number and in part by the number of houses in the street, she nevertheless feels that this is a desirable property for her house to have and decides that all her subsequent houses should exhibit it.

Write a program to find pairs of numbers that satisfy this condition. To start your list the first two pairs are: (house number, last number):

```
     6         8
    35        49
```

Input and Output

There is no input for this program. Output will consist of 10 lines each containing a pair of numbers, each printed right justified in a field of width 10 (as shown above).

902 - Password Search

Time limit: 3.000 seconds

Being able to send encoded messages during World War II was very important to the Allies. The messages were always sent after being encoded with a known password. Having a fixed password was of course insecure, thus there was a need to change it frequently. However, a mechanism was necessary to send the new password. One of the mathematicians working in the cryptographic team had a clever idea that was to send the password hidden within the message itself. The interesting point was that the receiver of the message only had to know the size of the password and then search for the password within the received text.

A password with size N can be found by searching the text for the most frequent substring with N characters. After finding the password, all the substrings that coincide with the password are removed from the encoded text. Now, the password can be used to decode the message.

Problem

Your mission has been simplified as you are only requested to write a program that, given the size of the password and the encoded message, determines the password following the strategy given above.

To illustrate your task, consider the following example in which the password size is three (N=3) and the text message is just `baababacb`. The password would then be `aba` because this is the substring with size 3 that appears most often in the whole text (it appears twice) while the other six different substrings appear only once (`baa` ; `aab` ; `bab` ; `bac` ; `acb`).

Input

The input file contains several test cases, each of them consists of one line with the size of the password, $0 < N \leq 10$, followed by the text representing the encoded message. To simplify things, you can assume that the text only includes lower case letters.

Output

For each test case, your program should print as output a line with the password string.

Sample Input

```
3 baababacb
```

Sample Output

```
aba
```

10341 - Solve It

Time limit: 3.000 seconds

Solve the equation:

$p*e^{-x} + q*\sin(x) + r*\cos(x) + s*\tan(x) + t*x^2 + u = 0$

where $0 <= x <= 1$.

Input

Input consists of multiple test cases and terminated by an EOF. Each test case consists of 6 integers in a single line: p, q, r, s, t and u (where $0 <= p,r <= 20$ and $-20 <= q,s,t <= 0$). There will be maximum 2100 lines in the input file.

Output

For each set of input, there should be a line containing the value of x, correct upto 4 decimal places, or the string "`No solution`", whichever is applicable.

Sample Input

```
0 0 0 0 -2 1
1 0 0 0 -1 2
1 -1 1 -1 -1 1
```

Sample Output

```
0.7071
No solution
0.7554
```

438 - The Circumference of the Circle

Time limit: 3.000 seconds

To calculate the circumference of a circle seems to be an easy task - provided you know its diameter. But what if you don't?

You are given the cartesian coordinates of three non-collinear points in the plane.

Your job is to calculate the circumference of the unique circle that intersects all three points.

Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing six real numbers $x_1, y_1, x_2, y_2, x_3, y_3$, representing the coordinates of the three points. The diameter of the circle determined by the three points will never exceed a million. Input is terminated by end of file.

Output Specification

For each test case, print one line containing one real number telling the circumference of the circle determined by the three points. The circumference is to be printed *accurately rounded* to two decimals. The value of π is approximately 3.141592653589793.

Sample Input

```
0.0 -0.5 0.5 0.0 0.0 0.5
0.0 0.0 0.0 1.0 1.0 1.0
5.0 5.0 5.0 7.0 4.0 6.0
0.0 0.0 -1.0 7.0 7.0 7.0
50.0 50.0 50.0 70.0 40.0 60.0
0.0 0.0 10.0 0.0 20.0 1.0
0.0 -500000.0 500000.0 0.0 0.0 500000.0
```

Sample Output

```
3.14
4.44
6.28
31.42
62.83
632.24
3141592.65
```

706 - LCD Display

Time limit: 3.000 seconds

A friend of you has just bought a new computer. Until now, the most powerful computer he ever used has been a pocket calculator. Now, looking at his new computer, he is a bit disappointed, because he liked the LC-display of his calculator so much. So you decide to write a program that displays numbers in an LC-display-like style on his computer.

Input

The input file contains several lines, one for each number to be displayed. Each line contains two integers s, n ($1 \leq s \leq 10, 0 \leq n \leq 99\,999\,999$), where n is the number to be displayed and s is the size in which it shall be displayed.

The input file will be terminated by a line containing two zeros. This line should not be processed.

Output

Output the numbers given in the input file in an LC-display-style using s ``-" signs for the horizontal segments and $s$ ``|" signs for the vertical ones. Each digit occupies exactly s+2 columns and 2s+3 rows. (Be sure to fill all the white space occupied by the digits with blanks, also for the last digit.) There has to be exactly one column of blanks between two digits.

Output a blank line after each number. (You will find a sample of each digit in the sample output.)

Sample Input

```
2 12345
3 67890
0 0
```

Sample Output

```
    --   --         --
   |    |    | |  | |
   |    |    | |  | |
    --   --   --   --
   | |       |    |    |
   | |       |    |    |
    --   --         --

 ---   ---   ---   ---   ---
|         | |   | |   | |   |
|         | |   | |   | |   |
|         | |   | |   | |   |
 ---         ---   ---
|   |     | |   |     | |   |
|   |     | |   |     | |   |
|   |     | |   |     | |   |
 ---         ---   ---   ---
```