Problem
Cipher

Input File: CipherIn.txt
Output File: CipherOut.txt
Project File: Cipher

Your little brother Ryan, as younger siblings often do, has been teasing you. To make things worse, he has also been reading your mail. For this reason you have nicknamed him “the beast”. You can tolerate the teasing but you can no longer allow “the beast” to read your mail; “the beast” must be tamed.

To control “the beast”, you and your friends have agreed on an offset cipher code to encrypt your mail. Naturally, “the beast” will never be given the cipher code, in which each letter in the original message will have a unique replacement character. The replacement character will be the character in the ASCII table at a given integer offset from the original character. Thus, if the offset is –1, the character ‘b’ will be replaced with the character ‘a’.

You are to write a program to generate the encrypted messages. Spaces and new lines will not be encrypted, and offsets will be chosen so that only the ASCII printable characters will appear in the encrypted message.

Inputs
The first line of the file will contain the integer offset. Subsequent lines will contain the message to be encrypted.

Outputs
The encrypted message containing the same number of lines as the un-encrypted message.

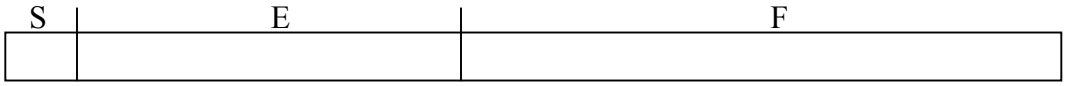
Sample input
```
-10
Hi Breanne,

How are you today? I do not think the beast
will be able to read this message.
Your Grandfather is such a nice person and a great skier.

Love,
Tom
```

Sample output
```
>_ 8h[Wdd["

>em Wh[ oek jeZWo5 ? Ze dej j^_da j^[ X[Wij
m_bb X[ WXb[ je h[WZ j^_i c[iiW][$
Oekh =hWdZ\Wj^[h _i ikY^ W d_Y[ f[hied WdZ W ]h[Wj ia_[h$

Bel["
Jec
```

Problem
Floats

Input File: FloatsIn.txt
Output File: FloatsOut.txt
Project File: Floats

You are writing the part of a compiler that converts floating-point numbers, stored in memory, to base 10 representations. Floating-point numbers are represented in memory using an IEEE standard representation. Under this standard, 32 bits are used to store one floating point value. These thirty-two bits are divided into three fields (groups of bits) as shown below:



Field S is one bit wide, field E is 8 bits wide, and F is 23 bits wide. S and E are binary unsigned integers. F is a binary fraction. The base 10 value of the stored floating point number is calculated as:

$$(-1)^S * 1.F * 2^{(E-128)}$$

Thus, if S were 1, E were $10000001 = 129_{10}$ and the three left most bits of F were on, F would be ½ + ¼ + $^1/_8$ = $.875_{10}$ and the base 10 floating point number stored would be:

$$-3.75 = (-1)^1 * 1.875 * 2^{129-128}$$

Write a program to translate the internal representation of floating point numbers stored using the IEEE standard into their base 10 representation.

Inputs
The 32-bit binary representation of the floating point numbers, one per line. Each output should have a precision of 12.

Outputs
The base 10 representation of the floating point numbers, one per line.

Sample inputs
11000000111100000000000000000000

00111111101010100000000000000000

01000001010101010101000000000000

Sample outputs
-3.750000000000

0.332031250000

13.332031250000

Problem
Math

Input File: MathIn.txt
Output File: MathOut.txt
Project File: Math

Mathematicians on the planet Earth, write math expressions using in-fixed notation. In this notation, math operators are written between the operands they operate on (e.g., 2 + 3). On Mars, math strings are written in post-fixed form. In this notation, math operators are written after the two operands they operate on (e.g., 2 3 +).

The nice thing about post-fixed notation is that we don't need rules of precedence to decide what math operator should be evaluated first. For instance, in the in-fixed math string 6 + 4 / 2, the rules of precedence dictate that we should divide before we add. Without these rules, there is an ambiguity in the expression. The same math expression written in post fixed notation is 4 2 / 6 +.

Fortunately programmers who write translators are from Mars, and they translate math expressions from in-fixed to post-fixed notation before evaluating them. Thus, we need not worry about the rules of precedence at run time.

To evaluate a post-fixed string, we start at the left most character and examine characters until we find an operator. Once an operator is found, it is applied to the two operands immediately before it, and then the operand and the two operators in the post-fixed string are replaced with the result. Then we continue from this point, repeating the procedure. When we reach the end of the string, there will only be one item left in the string, the result. Thus the in-fixed string 5 6 2 + 4 / - is equivalent to the post-fixed string 5 - (6 + 2) / 4, both of which evaluate to 3.

Inputs
The input file will contain math strings in post-fixed notation, one per line. Operands will consist of one digit. Operators and operands will be separated by one space. There will be no more than 80 characters in the math expressions.

Outputs
There will be one line of output for each math expression. The line will contain the value of the math expression.

Sample input
1 5 9 + 8 – +
5 6 2 + 4 / -
4 7 9 8 * + 2 + -

Sample Output
7
3
77

Problem
Math

Input File: MathIn.txt
Output File: MathOut.txt
Project File: Math

Mathematicians on the planet earth, write math expressions in-fixed notation. In this notation, math operators are written between the operands they operate on (e.g. 2 + 3). On Mars, math strings are written in post-fixed form. In this notation math operators are written after the two operands they operate on (e.g. 2 3 +).

The nice thing about post fixed notation, is that we don't need rules of precedence to decide what math operator should be evaluated first. For instance, in the in-fixed math string: 6 + 4 / 2 the rules of precedence dictate that we should divide before we add. Without these rules, there is an ambiguity in the expression. The same math expression written in post fixed notation is 4 2 / 6 +.

Fortunately programmers that write translators are from mars and they translate math expression from infixed to post fixed notation before evaluating them. Thus, we need not worry about the rules of precedence at run time.

To evaluate a post-fixed string, we start at the right most character and examine characters until we find an operator. Once an operator is found it is applied to the two operands immediately before it and then the operand and the two operators are replaced in the string with the result. Then we continue from this point, repeating the procedure. When we reach the end of the string, there will only be one item left in the string, the result. Thus the in-fixed string: 5 6 2 + 4 / - is equivalent to the post-fixed string: 5 - (6 + 2) / 4 both of which evaluate to 3.

Inputs
The input file will contain math strings in post-fixed notation, one per line. Operands will consist of one digit. Operators and operands will be separated by one space. There will be no more than 80 characters in the math expressions.

Outputs
There will be one line of output for each math expression. The line will contain the value of the math expression.

Sample input
5 9 + 8 –
5 6 2 + 4 / -
7 9 8 * + 2 +

Sample Output
7
3
73

Problem
Primes

Input File: PrimesIn.txt
Output File: PrimesOut.txt
Project File: Primes

Your cousin Geoff is having trouble in his math class. He is studying prime numbers, and a special subset of the prime numbers called 4k+3 primes. A prime number is an integer greater than 1 that is only evenly divisible by itself and 1. A prime, p, is a 4k+3 prime if there is some integer k such that $p = 4k+3$.

Ms. Maggie, Geoff's math teacher, has asked him to find the lowest prime number above a given integer, n. Geoff is stumped, and frankly he suspects Ms. Maggie can't find the 4k+3 prime either. Your job is to help cousin Geoff and Ms. Maggie by writing a program to determine the lowest 4k+3 prime above an integer, n.

Inputs
The input file will contain several values of n, one per line.

Outputs
For each value of n, output the corresponding value of p (the lowest 4k+3 prime above n) one output per line.

Sample input
14
278
1145
2784

Sample output
19
283
1151
2791

Problem
Ships

Input File: ShipsIn.txt
Output File: ShipsOut.txt
Project File: Ships

Admiral Billy conducts night training exercises for his fleet to simulate close-quarter engagements that might occur just outside a port in a relatively confined area. One of these exercises involves a large number of ships (up to 20) trying to rush for the open sea from various points in a bay while running in "cloaked" mode (no lights with low engine power).

The likelihood that two ships will collide under Admiral Billy's direction is extremely high. Therefore, the Navy has decided to treat the bay area used in the exercise as a two-dimensional x-y plane. A Global Positioning System (GPS) will identify the location of each participating ship as an ordered pair (x, y) in the x-y plane. You are to help by writing a program that takes the collection of ship positions and identifies the two ships that are the closest to each other. Then admiral Billy can then issue a warning to each vessel to modify its course (and thereby retain his pension). You may assume that only one pair of ships will need to be re-directed by Admiral Billy.

Inputs

The input file contains an unspecified number of fleet position specifications. Each specification begins with a line containing a single integer value N, representing the number of ships currently in the fleet ($2 \le N \le 20$). The subsequent N lines complete the specification by listing a coordinate location (x. y) of each ship in the fleet, one per line, from Ship 1 to Ship N.

Outputs

There will be one output line per specification. The line will identity the two closest ships annotated as:

Ships x and y are the closest pair.

x and y being the ship numbers, with $x < y$.

(Sample inputs and outputs are on the next page)

Sample input

```
7
0.7 2
1 2
2 2
1 1.5
2 1
3 1
4 2
8
0.25 1
1.1 0.5
0.5 0.5
0.6 0.8
1 1
2 1
2 0.7
0.9 0.5
```

Sample output

```
Ships 1 and 2 are the closest pair.
Ships 2 and 8 are the closest pair.
```

Problem
Survivor

Input File: SuvivorIn.txt
Output File: SuvivorOut.txt
Project File: Survivor

The latest reality-TV gimmick has stranded *n* survivors and a referee monkey, Zimba, on Grand Cayman Island. The survivors decide to gather all of the *m* coconuts on the island into one pile. That night while they are sleeping, the first survivor wakes up and decides to take his portion of the coconuts. He divides the coconuts into *n* equal piles of whole coconuts (leaving perhaps a few remaining) and hides one of the piles. He puts the other *n-1* piles (and the remainders) back into one large pile and buys Zimba's silence with one of the coconuts from the large pile. The first survivor then goes back to sleep. Each of the other survivors in turn wakes up and follows the same routine. Write a program to determine how many coconuts remain after all of the survivors are done with their late night chicanery.

Inputs

The input file will consist of a sequence of integers *n* and *m*, one number per line. The input will terminate with *n*=0 and *m*=0.

Outputs

Your program should output one line, for each *n* and *m* pair, that gives the number of coconuts remaining, with an appropriate label (see below).

Sample input

```
3
106
5
14
10
3000
0
0
```

Sample output

```
The number of coconuts remaining is 30
The number of coconuts remaining is 3
The number of coconuts remaining is 1043
```