A good way to transmit information over an unreliable connection is to use redundancy to eliminate the damage caused by small errors. For instance, a checksum on data can be used to reliably determine if a mistake exists anywhere in the data. And if one byte (anywhere in the data) is missing, the checksum can be used to reconstruct the value of that byte. We will examine a generalized redundancy approach that allows us to reconstruct a message from which some information has been lost.

Let p be a prime. Suppose we have sequence of p "chunks", $a_1, a_2, \ldots, a_p$, each with a value between 0 and $p-1$ inclusive. We can construct a "difference sequence," $b_1, b_2, \ldots, b_p$, which has the same format, and contains the differences between successive elements of a. (Modular arithmetic is used to keep the values between 0 and $p-1$) So $b_1 \equiv a_2 - a_1 (\bmod\ p)$; $b_2 \equiv a_3 - a_2 (\bmod\ p)$; and so on, wrapping around to give $b_p \equiv a_1 - a_p (\bmod\ p)$.

We can take the difference sequence of this difference sequence to get a "2-nd difference sequence". In general, let the "n-th difference sequence" be the result of applying this operation to a sequence n times.

Not every sequence can be an "n-th difference sequence", so such a sequence is redundant in a very useful way. If somebody tells you they have a message (a sequence of this form) that IS an n-th difference sequence, and sends it to you, you can reconstruct their message even if up to n "chunks" are lost along the way. It is straightforward to encode useful information this way: you can write $p-n$ "chunks" of information, then add n "chunks" to make the sequence an n-th difference sequence (of some unimportant initial sequence).

Write a program to reconstruct messages of this form.

Input

There will be up to 100 cases, each consisting of two lines. The first line contains p, a prime number less than 100, and n, a positive integer. The second line contains p integers from 0 to $p-1$, containing the message; however, up to n of them may be missing, and will be replaced by question marks.

Input is terminated by a line containing two zeros.

Output

For each case, output one line containing the reconstructed message, which will be an "n-th difference sequence". If this is impossible, output '`Invalid message!`'.

Sample Input

```
5 1
1 3 4 ? 3
5 1
1 3 4 2 3
0 0
```

Sample Output

```
1 3 4 4 3
Invalid message!
```