Problem 4—The Hamming Code

Those darn Romulans didn't fall for the Corbomite Maneuver this time, and they've broken Code Two, so what is the Enterprise to do? They need to send a message to Starfleet Command right now and they're concerned that the Romulans might interfere with the transmission. However, Chief Engineer Montgomery Scott has guaranteed that the Romulans will not be able to do any more damage to the transmission than to flip a single bit, and they might not be able to do even that. Mr. Scott has a plan to correct the error, though; he will use a Hamming Code. Imagine each bit is numbered from *right-to-left* starting from 1. All the bits whose positions are powers of 2 (1, 2, 4, 8, etc.) are *not* part of the original message, but are insertions into the bit pattern. These are called parity bits and each parity bit marks a certain portion of the message. Each parity bit marks those bits whose position number, written in binary, contains a one in the position corresponding to that parity bit.

For example, 23, written in binary is 10111. The 1-bit, 2-bit, 4-bit, and 16-bit are all 1 in the binary representation of 23, so parity bits 1, 2, 4, and 16 all mark the bit in position 23 of the transmission. Notice that all parity bits mark themselves and no parity bit marks any other.

Each parity bit is chosen to be zero or one so that the total number of ones in the positions marked by that parity bit (including itself) is even. For example, consider the three-bit message 1 0 1. With parity bits inserted into the message, we have 1 0 _ 1 _ _. The 1-bit will mark positions 1, 3, and 5, and so has to be 1. The 2-bit will mark positions 2, 3, and 6, and so must be 0. The 4-bit will mark positions 4, 5, and 6, and so must be 1. So, the transmitted message will be 1 0 1 1 0 1.

Should one bit be flipped during transmission, this bit can be corrected by adding up the parity positions that are now wrong. Suppose, the transmission 1 1 1 1 0 1 was received. The 1-bit is wrong since it now marks an odd number of ones. The 2-bit is still correct, and the 4-bit is also wrong. So, 1+4 = 5, so it must be the 5-bit that flipped. The transmission should have been 1 0 1 1 0 1, and when the parity bits are stripped out, we see that the original message was 1 0 1. If none of the parity bits are wrong in the received transmission, then it is safe to assume that the transmission occurred without error at all.

Given a received transmission with at most one error, you are to correct the error, strip the parity bits, and print the original transmission.

INPUT SPECIFICATION. You will be a given a set of lines consisting of an arbitrary number of 0's and 1's in any order, of length at least 3, each followed by **<EOLN>**. The last input case will be followed by "*END*<**EOLN**>".

OUTPUT SPECIFICATION. The output cases should appear in the same order as the input cases. Each output case should be in the form "Case *c*: *b*" where *c* is the case number and *b* is the original bit string. Each output case should be followed by two <**EOLN**>'s.

SAMPLE INPUT.

```
101101<EOLN>
111101<EOLN>
*END*<EOLN>
<EOF>
```

SAMPLE·OUTPUT.

```
Case·1:·101<EOLN>
<EOLN>
Case·2:·101<EOLN>
<EOLN>
<EOF>
```