All-Star Contest
MORE PARITY

PROBLEM: If a binary number contains an even number of 1 bits, we say that it has *even parity*. If the number contains an odd number of 1 bits, it has *odd parity*. When data must be transmitted from one device to another, there is always the possibility that an error might occur. Detection of a single incorrect bit in a data word can be detected simply by adding an additional *parity bit* to the end of the word. If both the sender and receiver agree to use even parity, for example, the sender can set the parity bit to either 1 or zero so as to make the total number of 1 bits in the word an even number:

8-bit data value: 1 0 1 1 0 1 0 1
added parity bit: 1
transmitted data: 1 0 1 1 0 1 0 1 1

8-bit data value: 1 0 1 1 0 1 0 0
added parity bit: 0
transmitted data: 1 0 1 1 0 1 0 0 0

The receiver of a transmission also counts the 1 bits in the received value, and if the count is not even, an error condition is signaled and the sender is usually instructed to re-send the data. In 1950, Richard Hamming developed an innovative way of adding bits to a number in such a way that transmission errors involving no more than a single bit could be detected and corrected. For data of size 2^n bits, n+1 parity bits are embedded to form a codeword. The parity bit positions are powers of 2: {1,2,4,8,16,32...}. All remaining positions hold data bits. The 16-bit data value 1000111100110101 would be stored as the 21-bit codeword as follows:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
P	P	1	P	0	0	0	P	1	1	1	1	0	0	1	P	1	0	1	0	1

Parity Bit	Rule
1	use 1, skip 1, use 1, skip 1, ...
2	use 2, skip 2, use 2, skip 2, ...
4	use 4, skip 4, use 4, ...
8	use 8, skip 8, use 8, ...

The adjacent table gives the rules for calculating the value of each parity bit up to bit 8.

Now it's time to put all of this information together and create a codeword. For this example we will use even parity and the 8-bit data value 1 1 0 0 1 1 1 1, which will produce a 12-bit codeword. Let's start by filling in the data bits:

1	2	3	4	5	6	7	8	9	10	11	12
P	P	**1**	P	**1**	**0**	**0**	P	**1**	**1**	**1**	**1**

To calculate the parity bit in position 1, start with the parity bit location, which is initially 0, and sum the bits in positions 1, 3, 5, 7, 9, and 11: (0+1+1+0+1+1 = 4). This is done by the rule: use 1, skip 1. The positions 1,3,5,7,9 and 11 are called the *bit group* for P1. This sum is even so parity bit 1 should be assigned a value of 0.

1	2	3	4	5	6	7	8	9	10	11	12
0	P	**1**	P	**1**	**0**	**0**	P	**1**	**1**	**1**	**1**

To generate the parity bit in position 2, sum the bits in positions 2, 3, 6, 7, 10, and 11: (0+1+0+0+1+1 = 3). This was done by the rule: use 2, skip 2. The sum is odd, so we assign a value of 1 to parity bit 2. This produces even parity for the combined group of bits 2, 3, 6, 7, 10, and 11:

1	2	3	4	5	6	7	8	9	10	11	12
0	**1**	**1**	P	**1**	**0**	**0**	P	**1**	**1**	**1**	**1**

When all parity bits have been created, the resulting codeword is: **01**1**0**100**0**1111.

When a codeword is received, the receiver must verify the correctness of the data. This is accomplished by counting the 1 bits in each bit group (mentioned earlier) and verifying that each has the agreed parity. Here are the bit groups for a 12-bit code value:

Parity Bit	**Bit Group**
1	1, 3, 5, 7, 9, 11
2	2, 3, 6, 7, 10, 11
4	4, 5, 6, 7, 12
8	8, 9, 10, 11, 12

If parity is even and one of these groups produces an odd numbered sum, the receiver knows that a transmission error occurred. As long as only a single bit was altered, it can be corrected. The method can be best shown using concrete examples.

Example 1: Suppose that the bit in position 4 was reversed, producing 011**1**10001111. The receiver would detect an odd parity in the bit group associated with parity bit 4. Since the bit in position 4 appears in no other bit group, the receiver would toggle this bit, thus correcting the transmission error.

Example 2: Suppose that bit 7 was reversed, producing 011010**1**01111. The bit groups based on parity bits 1, 2, and 4 would have odd parity. The only bit that is shared by all three groups (the *intersection* of the three sets of bits) is bit 7, so again the error bit is identified. See the chart below:

Parity Bit	Bit Group
1	1, 3, 5, **7**, 9, 11
2	2, 3, 6, **7,** 10, 11
4	4, 5, 6, **7**, 12
8	8, 9, 10, 11, 12

Example 3: Suppose that bit 6 was reversed, producing 011011001111. The groups based on parity bits 2 and 4 would have odd parity. Notice that two bits are shared by these two groups (their intersection): 6 and 7. See the chart below:

Parity Bit	Bit Group
1	1, 3, 5, **7**, 9, 11
2	2, 3**, 6, 7**, 10, 11
4	4, 5, **6, 7**, 12
8	8, 9, 10, 11, 12

But notice that the 7 occurs in group 1, which has even parity. This leaves bit 6 as the only choice as the incorrect bit.

INPUT: There will be 10 lines of input. Each line will contain 2 strings. The first string will be a hexadecimal value that needs to be converted to binary to form the codeword. The maximum length of a codeword will be 132 bits. The second string will be either the word ODD or EVEN. This second string gives the original parity of the transmitted value.

OUTPUT: For each line of input, print the position of the parity error. ACSL guarantees that there will be at most one error per input string. If there is no error, then print the word NONE.

SAMPLE INPUT	SAMPLE OUTPUT
1. E8F, EVEN	1. 1
2. 7222F, EVEN	2. 2
3. B9CD66, ODD	3. 4
4. 912, EVEN	4. 6