Source File: `~/1337/31/lab31.(C|CPP|cpp|c++|cc|cxx|cp)`
Input: Standard Input
Output: Standard Output
Value: 4

The Hamming Character Code is a code in which single-bit errors can be detected and corrected. Each 7-bit ASCII character is embedded in an 11-bit code word called a ***Hamming character***. Bit positions of the hamming character are numbered from most significant to least significant. The encoding is done as follows:

bit positions:	1	2	3	4	5	6	7	8	9	10	11
bits:	p	p	d	p	d	d	d	p	d	d	d

Bits 1, 2, 4, and 8, all having bit positions that are a power of two, are ***parity bits***. Bits 3, 5, 6, 7, 9, 10, and 11 are the ***data bits*** which represent the most to least significant bits of the embedded ASCII character. The parity bits are computed so that

$$\text{bit } 1 = (\text{bit } 3 + \text{bit } 5 + \text{bit } 7 + \text{bit } 9 + \text{bit } 11) \bmod 2$$
$$\text{bit } 2 = (\text{bit } 3 + \text{bit } 6 + \text{bit } 7 + \text{bit } 10 + \text{bit } 11) \bmod 2$$
$$\text{bit } 4 = (\text{bit } 5 + \text{bit } 6 + \text{bit } 7) \bmod 2$$
$$\text{bit } 8 = (\text{bit } 9 + \text{bit } 10 + \text{bit } 11) \bmod 2$$

For example, the ASCII character 'M' having 7-bit code 1001101 will be represented by 01110010101.

Suppose the text encoded as a sequence of Hamming characters is transmitted through a channel that can introduce at most one bit error per Hamming character. We may determine the location of the error and correct it as follows. If the Hamming character 01110010101 representing 'M' were to be received as 01110010001, the location of the errant bit may be computed magically as the sum of the positions of the parity bits that disagree with their recomputed values. The offending bit is then complemented to determine the correct Hamming character. Thus the received word 01110010001

$*$ bit 1 is 0 $\quad 1 = (\text{bit } 3 + \text{bit } 5 + \text{bit } 7 + \text{bit } 9 + \text{bit } 11) \bmod 2$
bit 2 is 1 $\quad 1 = (\text{bit } 3 + \text{bit } 6 + \text{bit } 7 + \text{bit } 10 + \text{bit } 11) \bmod 2$
bit 4 is 1 $\quad 1 = (\text{bit } 5 + \text{bit } 6 + \text{bit } 7) \bmod 2$
$*$ bit 8 is 0 $\quad 1 = (\text{bit } 9 + \text{bit } 10 + \text{bit } 11) \bmod 2$

$1 + 8 = 9$, which indicates the location of the offending bit! If bit 9 in 01110010001 is changed, it yields the corrected Hamming character for 'M', 01110010101. This decoding method works for all Hamming characters with one bit error. Naturally, if there are no bit errors, there will be no discrepancies between parity bits and their recomputed values.

Input

A message is recorded in the input text file as a sequence of ***short*** integers, one per line, each representing a single Hamming character with at most one bit error. For example the Hamming character that corresponds to ASCII 'M' is represented by 917. Write a program that decodes the message, correcting for single-bit errors. The input should be read from standard input (use redirection). The message is of unknown length; thus, your program should process integers until the end-of-data marker is encountered.

Output

The decoded message should be written to the standard output as characters, not integers.

A sample execution sequence is shown in Figure 1 and an explanation of the input is shown in Figure 2. To use the `Makefile` as distributed in class, add a target of `lab31` to `targets1srcfile`.

```
1  newuser@csunix ~> cd 1337
2  newuser@csunix ~/1337> mkdir 31
3  newuser@csunix ~/1337> cd 31
4  newuser@csunix ~/1337/31> cp /usr/local/1337/data/31/* .
5  newuser@csunix ~/1337/31> cp /usr/local/1337/src/Makefile .
6  newuser@csunix ~/1337/31> touch lab31.cpp
7  newuser@csunix ~/1337/31> # Edit Makefile and lab31.cpp
8  newuser@csunix ~/1337/31> make lab31
9  g++ -g -Wall -std=c++11 -c lab31.cpp -I/usr/local/1337/include -I.
10 g++ -o lab31 lab31.o -L/usr/local/1337/lib -lm -lbits
11 newuser@csunix ~/1337/31> cat 01.dat
12 22992
13 3533
14 -20667
15 24407
16 14937
17 -17578
18 23535
19 9370
20 newuser@csunix ~/1337/31> cat 01.dat | ./lab31
21 Your Name - CS 1337 - Lab 31
22
23 Hamming
24 newuser@csunix ~/1337/31> cat 01.dat | ./lab31 > my.out
25 newuser@csunix ~/1337/31> diff 01.out my.out
26 newuser@csunix ~/1337/31> cat 02.dat | ./lab31 > my.out
27 newuser@csunix ~/1337/31> diff 02.out my.out
28 newuser@csunix ~/1337/31>
```

Figure 1. Commands to Compile, Link, & Run Lab 31

Input Integer	Input Integer in Hexadecimal	Low-Order 11 Bits	Offending Bit	Resulting Character
22992	59D0	00111010000	5	1001000 = H
3533	0DCD	10111001101	9	1100001 = a
-20667	AF45	11101000101	7	1101101 = m
24407	5F57	11101010111	10	1101101 = m
14937	3A59	01001011001	3	1101001 = i
-17578	BB56	01101010110	0	1101110 = n
23535	5BEF	01111101111	6	1100111 = g
9370	249A	10010011010	0	0001010 = \n

Figure 2. Explanation of Input