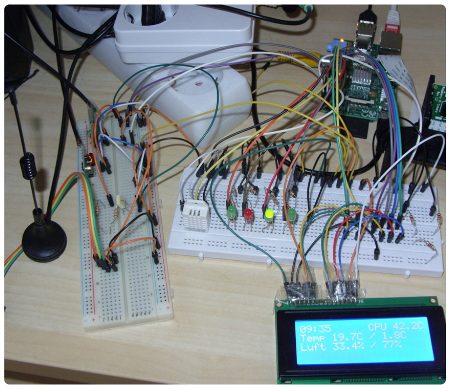
Jumbled Communication

Problem ID: communication
CPU Time limit: 2 seconds
Memory limit: 1024 MB
Difficulty: 2.0

Author: Gregor Behnke
Source: Northwestern Europ
Regional Contest (NWERC) 20
License: BY-SA

Your best friend Adam has recently bought a Raspberry Pi and some equipment, including a wireless temperature sensor and a 433MHz receiver to receive the signals the sensors sends. Adam plans to use the Raspberry Pi as an in-door display for his weather sensor. As he is very good with electronics, he quickly managed to get the receiver to receive the signals of the sensor. However, when he looked at the bytes sent by the sensor he could not make heads or tails of them. After some hours looking through a lot of websites, he found a document explaining that his weather sensor scrambles the data it sends, to prevent it from being used together with products from other manufacturers.



© *NWERC Jury*

Luckily, the document also describes how the sensor scrambles its communication. The document states that the sensor applies the expression `x ^ (x << 1)` to every byte sent. The `^` operator is bit-wise XOR[1], e.g., 10110000 ˆ 01100100 = 11010100. The `<<` operator is a (non-circular) left shift of a byte value[2], e.g., 10111001 << 1 = 01110010.

In order for Adam's Raspberry Pi to correctly interpret the bytes sent by the weather sensor, the transmission needs to be unscrambled. However, Adam is not good at programming (actually he is a pretty bad programmer). So he asked you to help him and as a good friend, you are always happy to oblige. Can you help Adam by implementing the unscrambling algorithm?

Input

The input consists of:

- one line with an integer n ($1 \leq n \leq 10^5$), the number of bytes in the message sent by the weather sensor;
- one line with n integers $b_1, \ldots, b_n$ ($0 \leq b_i \leq 255$ for all i), the byte values of the message.

Output

Output n byte values (in decimal encoding), the unscrambled message.

Sample Input 1

```
5
58 89 205 20 198
```

Sample Output 1

```
22 55 187 12 66
```

Footnotes

1. In bit-wise XOR, the ith bit of the result is 1 if and only if exactly one of the two arguments has the ith bit set.
2. In `x << j`, the bits of x are moved j steps to the left. The j most significant bits of x are discarded, and j zeroes are added as the least significant bits of the result.