Big- or Little- Endian

All computer memory is organized into 8-bit bytes. For integer values that require a type with more than 8-bits, such as the typical 2-byte, 4-byte, and 8-byte integral types available on most modern hardware, there is no universal agreement as to how to order those bytes and two incompatible storage schemes exist.

The first stores integers as groups of consecutive 8-bit bytes with the least significant byte occupying the lowest memory address within the group and the most significant byte occupying the highest memory address.

The second is just the reverse; the least significant byte is stored in the highest memory address within the group, and the most significant byte is stored in the lowest memory address.

Those schemes have been dubbed Little Endian and Big Endian, respectively.

We assume that signed integers are stored using "“"two’s complement" representation.

When binary integer data is shared between a Little Endian and Big Endian machine, a data conversion must be performed which involves reversing the bytes of the data. Once the bytes have been reversed the integer is then correctly interpreted by the hardware as the original value from the opposite-endian machine. Your task is to write a program that will read a list of integers and report the integers that the binary representations of the input integers would represent on an opposite-endian machine.

Input

A number `n` , `-2147483648 <= n <= 2147483647` .

Output

The number `n` followed by the phrase "converts to" followed by the other endian-value (single space between the three parts).

Example 1

```
Input:
123456789
Output:
123456789 converts to 365779719
```

Example 2

```
Input:
-123456789
Output:
-123456789 converts to -349002504
```

Example 3

```
Input:
1
Output:
1 converts to 16777216
```

Example 4

```
Input:
16777216
Output:
16777216 converts to 1
```