One Little, Two Little, Three Little Endians

Writing programs that are completely portable across different operating systems, operating system versions and hardware platforms is a challenging task. One of the difficulties encountered is a result of decisions made by hardware manufacturers about how they will store integer data in memory. Because these representations can differ from machine to machine, sharing binary data often cannot be done without modifying the way in which the data is stored or the way in which it is handled by one or more of the platforms.

Fortunately there is near-universal agreement among hardware manufacturers that addressable memory be ordered into 8-bit bytes. For integer data values that require more than 8-bits, such as the typical 2- byte, 4-byte, and 8-byte integer types available on most modern hardware, there is no such agreement 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 location within the group and the most significant byte occupying the highest memory location. The second is just the reverse; the least significant byte is stored in the highest memory location within the group, and the most significant byte is stored in the lowest memory location. The computing industry has dubbed these schemes Little Endian and Big Endian, respectively. There is also near-universal agreement that signed integers are stored using "two's complement" representation, and you may assume that this is the case.

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. The object of this problem 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

The input will consist of a list integers. The end of the input file marks the end of the list. All input integers can be represented as a 32-bit signed integer value. That is, the input integers will be in the range -2147483648 to 2147483647.

Output

For each input integer a single line should be printed to the output file. The line should contain the input integer followed by the phrase ``converts to" followed by one space followed the other-endian value.

Sample Input

```
123456789
-123456789
1
16777216
20034556
```

Sample Output

```
123456789 converts to 365779719
-123456789 converts to -349002504
1 converts to 16777216
16777216 converts to 1
20034556 converts to -55365375
```

Miguel A. Revilla
1999-01-11