Q2. The Bits in a Loop! (20%)

We know that the rightmost bits will be dropped if we use the bitwise operator >>. However, it is possible to create a loop of bits in which the dropped leftmost or rightmost bits can appear at the rightmost and leftmost sides! Let's see the example.

Assume we have an 8-bit number that looks like the figure below (the decimal value of this number is 27).

0	0	0	1	1	0	1	1

We shift this number to the right by 2 places. Instead of dropping the two rightmost bits (11), we make these two bits loop back to the leftmost side. The outcome looks like the below figure (the decimal value is 198).

1	1	0	0	0	1	1	0

Although we use an 8-bit number in the above example, you will need to deal with the `unsigned short int` (16 bits) number in your program. Hint: the conversion specification for `unsigned short int` is `%hu`.

- **Input Format**
  - The first line has an `unsigned short int` **n**: the number to be shifted, and the number will be input in a decimal form.
  - The second line has an integer **p**: the number of places to be shifted to the right.
- **Output Format**
  Print the decimal number of the `unsigned short int` **n** shifted by **p** places.
- **Technical specifications**
  - $0 \leq n \leq 2^{16} - 1$
  - $1 \leq p \leq 16$

Input Example	Output Example
2 1	1
12345 5	51585
47 2	49163
0 16	0

65535 16	65535