# Problem 5 – Bit Roller

Nakov takes a 19-bit number and rolls it on the right (moves its most right bit at the left most position). He tried this several times and he found it is too easy. Then he invented a more complex game: freeze a certain bit as a pillar and roll right all other bits several times. Now he is happy but he wants to automate this process.

Help Nakov to write a program that **rolls r times a 19-bit number n with a frozen bit at position f**.

Example: we have the number **n** = **2521**, which is **0000000100111011001** in binary (as a 19-bit number). We **freeze** the bit at position **f** = **8** (we count the positions from the right, starting from 0). We roll out the number **r** = **4** times. We obtain the result **295245** as follows:

* 2521(10) = 0000000100**1**11011001 🡪 1000000010**1**01101100 (roll right with frozen position 8)
* 1000000010**1**01101100 🡪 0100000001**1**00110110 (roll right with frozen position 8)
* 0100000001**1**00110110 🡪 0010000000**1**10011011 (roll right with frozen position 8)
* 0010000000**1**10011011 🡪 1001000000**1**01001101 = 295245(10) (roll right with frozen position 8)

### Input

The input data should be read from the console. It will consist of 3 lines:

* The number **n** stays at the first line.
* The number **f** stays at the second line.
* The number **r** stays at the third line.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print the obtained result after rolling **r** times **n** with a frozen bit at position **f** at the console (as decimal number).

### Constraints

* The number **n** will be a 19-bit unsigned **integer** (in the range [0…524287]).
* The number **f** will be integer in the range [0…18].
* The number **r** will be integer in the range [0…100].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2521  8  4 | 295245 | 2521(10) = 0000000100**1**11011001 🡪 1000000010**1**01101100 🡪 0100000001**1**00110110 🡪 0010000000**1**10011011 🡪 1001000000**1**01001101 = 295245(10) |
| 480678  18  2 | 447849 | 480679(10) = **1**110101010110100110 🡪 **1**011010101011010011 🡪 **1**101101010101101001 = 447849(10) |