# Problem 5 – Bit Shooter

We are given a **bit field** in the form of **64-bit integer number**. We **shoot it 3 times**. Each shoot has a **center** and a **size**. The shoot damages **size** bits around the shoot **center** (makes these bits 0). Finally, the bit field is split into left side (bits 63 … 32) and right side (bits 31 … 0). Write a program that calculates **how many bits survive** (have value 1) after the shoots in the **left** side and in the **right** side of the bit field. The bits are numbered as traditionally in programming: from right to left from 0 to 63. Note that if we shoot at the bit field border (e.g. center=63 and size=15), the damage is **partial** because some damaged bits are outside of the bit field.

### Input

The input data should be read from the console. It will consist of exactly 4 lines:

* At the first line you will have a 64-bit integer, corresponding to the bit field.
* At each of the next 3 lines we have 2 numbers: shoot **center** and shoot **size** – integers, split by a space.

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

### Output

The output should be printed on the console. It should consists of exactly 2 lines:

* The first line print "**left: …**" and the number of alive bits in the left side.
* The second line print "**right: …**" and the number of alive bits in the right side.

### Constraints

* The **bit field** will be a 64-bit integer in the range [0 … 18 446 744 073 709 551 615].
* The values for the **center** will be integers will be integers in range [0 … 63].
* The values for the **size** will be **odd** integers in range [1 … 99].
* Allowed working time for your program: 0.25 seconds.
* Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4227378815862876842  1 5  22 3  58 7 | left: 10  right: 11 |
| **Comments** | |
| 4227378815862876842(10) = 0011101010101010101010101010001010101010100001001010101010101010 🡪 001110101010101010101010101000101010101010000100101010101010**0000** 🡪 0011101010101010101010101010001010101010**000**001001010101010100000 🡪 00**0000000**0101010101010101010001010101010000001001010101010100000 🡪  00000000001010101010101010100010 | 10101010000001001010101010100000 🡪 left: 10; right: 11 | |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9223372036854775806  15 7  43 15  58 1 | left: 15  right: 24 |