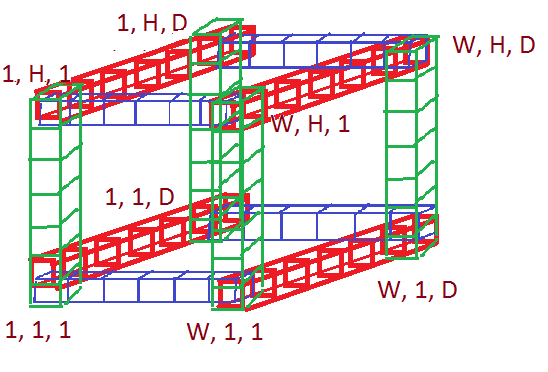
## Problem 3 – Laser

You are given a rectangular cuboid of size W (width), H (height) and D (depth) consisting of W \* H \* D cubes. All cubes have coordinates, corresponding to their position, starting from (1, 1, 1) – the coordinates of the "left, lower, near" cube and going to (W, H, D) – the coordinates of the "right, upper, far" cube.

Adjacent cubes we will call any cubes which share a common wall, edge or vertex. So, any cube in the cuboid has 26 (8 + 9 + 9) adjacent cubes, except if it is on the edge or wall of the cuboid (then it has less).

Let’s say we go inside one of the cubes in the cuboid (which is not on the side, or on the edge of the cube) and shoot a laser from it into one of the adjacent cubes. We can define the direction of the shot with a 3D vector, which has only 1, -1 or 0 as its coordinate values. So, for example if we want to shoot a laser straight up, that direction will be (0, 1, 0), if we want to shoot it straight down, the direction will be (0, -1, 0), if we want to shoot it forwards and up the direction will be (0, 1, 1) and if we want to shoot it left, forward and up the direction will be (-1, 1, 1). We have a total of 26 possible directions (the number of adjacent cubes).

When the laser passes through a cube, it burns it and it cannot pass through it again. Furthermore, if the laser reaches the wall of the cuboid (that is, a cube which is on the wall), it burns the cube there and reflects back into the cuboid, continuing to burn cubes it passes through. The reflection happens according to the laws of light reflection. Basically, the component of the direction vector, which would cause the laser to leave the cuboid, becomes the opposite number. So if a laser is moving in direction “right” (1, 0, 0), once it reaches the right wall of the cube it will reflect back, moving in direction “left (-1, 0, 0), but, in this case, that will cause it to visit an already burnt cube, so the laser will stop there.

Furthermore, all cubes on the edges (red green and blue cubes on the picture) of our cuboid are burned to begin with. So a laser will always stop before reaching the edge, because it cannot go into the edge cubes.

Write a program, which by given the width, height and depth of a cuboid, along with the coordinates of the cube from which the laser is shot and the direction of the shot, determines the last position the laser can reach in the cube.

Note: the starting position is burned during the shot, so it cannot be the answer (unless the laser travels into no other cubes, which can only happen if the next cube it should visit is an edge cube). The edges are burned by default, so no cube on the edge can be an answer either.

### Input

The input data should be read from the console.

On the first line of the input there will be the numbers W, H, D, separated by whitespaces – the width, height and depth of the cuboid

On the next line there will be the numbers startW, startH, startD, separated by whitespaces – the width, height and depth of the cube, from which the laser is shot.

On the third line there will be the numbers dirW, dirH, dirD, separated by whitespaces – defining the direction, in which the laser is shot.

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

### Output

The output data should be printed on the console.

On the only line of the output you should print exactly 3 numbers, separated by whitespaces – the width, height and depth of the last cube the laser will visit.

### Constraints

* W, H, D, startW, startH, startD, dirW, dirH and dirD will all be integers
* 4 < W, H, D < 100
* 1 < startW, startH, startD < W, H, D
* Each of the numbers dirW, dirH, dirD can only have the values -1, 1 or 0
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Sample input** | **Sample output** |
| **5 10 5**  **2 6 3**  **1 0 1** | **1 6 2** |
| **Explanation: Here we are only moving on the "sixth floor" of cubes (so we can meet only walls and no edges) and the movement looks like this:**  example | |