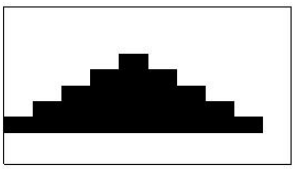
题目描述：

黑白图像常采用灰度图的方式存储，即图像的每个像素填充一个灰阶值，256 阶灰度图是一个灰阶值取值范围为 0-255 的灰阶矩阵，0 表示全黑、255 表示全白，范围内的其他值表示不同的灰度，比如下面的图像及其对应的灰阶矩阵：



255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255
255	255	255	255	0	255	255	255	255	255
255	255	255	0	0	0	255	255	255	255
255	255	0	0	0	0	0	255	255	255
255	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	0	255
255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255

但在计算机中实际存储时，会使用压缩算法，其中一种压缩格式和描述如下：
10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21
1、所有数值以空格分隔
2、前两个数分别表示矩阵的行数和列数
3、从第三个数开始，每两个数一组，每组第一个数是灰阶值，第二个数表示该灰阶值从左到右，从上到下（可理解为将二维数组按行存储在一维矩阵中）的连续像素个数。比如题目所述例子，“255 34”表示有连续34个像素的灰阶值是255。
如此，图像软件在打开此格式灰度图的时候，就可以根据此算法从压缩数据恢复出原始灰度图矩阵。
请从输入的压缩数恢复灰度图原始矩阵，并返回指定像素的灰阶值。

输入描述：10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21
3 4
输入包括两行，第一行是灰度图压缩数据，第二行表示一个像素位置的行号和列号，如：0 0 表示左上角像素。

输出描述：0
输入数据表示的灰阶矩阵的指定像素的灰阶值。

补充说明：1、系统保证输入的压缩数据是合法有效的，不会出现数据越界、数值不合法等无法恢复的场景；
2、系统保证输入的像素坐标是合法的，不会出现不在矩阵中的像素；
3、矩阵的行和列数范围为：(0,100]；
4、灰阶值取值范围：[0, 255]；

收起

示例1
输入：10 10 56 34 99 1 87 8 99 3 255 6 99 5 255 4 99 7 255 2 99 9 255 21
3 4
输出：99
说明：将压缩数据恢复后的灰阶矩阵第3行第4列的像素灰阶值是99.

示例2
输入：10 10 255 34 0 1 255 8 0 3 255 6 0 5 255 4 0 7 255 2 0 9 255 21
3 5
输出：255
说明：将压缩数据恢复后的灰阶矩阵第3行第5列的像素灰阶值是255.

```
#include <iostream>
#include <vector>
using namespace std;

int main(){
    int t;
    vector<int> list;
    while(cin >> t){
        list.push_back(t);
```

```
    }
    int n = list.size();
    int rows = list[0], colours = list[1];
    int r = list[n-2], c= list[n-1];
    int pos = r*colours +c+1,res = 0;
    for(int i = 2, sum = 0;i<n;i+=2){
        res = list[i];
        if(sum+list[i+1] >= pos) break;
        sum+=list[i+1];
    }
    cout <<res<<endl;
    return 0;
}
```