旋转矩阵

给定一个 N * N大小的方阵，顺时针旋转给定的度数，然后打印旋转后的矩阵。

输入描述

第一行给定正整数 N 和旋转的度数 x ，空格隔开，接下来有 N 行数据，每行 N 个数据，空格隔开。

矩阵的元素都是`int`类型，0 <= x <= 1000000 ，保证 x 是90的倍数

输出描述

按行输出旋转后的矩阵。

示例1

```
输入：
3 90
1 2 3
4 5 6
7 8 9
输出：
7 4 1
8 5 2
9 6 3
```

示例2

```
输入：
4 180
5 1 9 11
2 4 8 10
13 3 6 7
15 14 12 16
输出：
16 12 14 15
7 6 3 13
10 8 4 2
11 9 1 5
```

ArrayList

用数组实现一个简单的 ArrayList ， 数组元素的类型为`int`,支持如下操作

数组的初始大小为0，**在增加第一个元素时将数组大小设为10。**数组填满之后再增加元素时需要进行扩容，按照1.5倍扩容（建议使用`oldCapacity + (oldCapacity / 2)`或`oldCapacity + (oldCapacity >> 1)`），数组容量不能减小。

> add x: 在数组的末尾增加x
>
> remove x: 删除**第一个**值为x的元素，如果数组中包含多个x，只删除第一个，后面元素往前移动；有可能删除的数在数组中并不存在
>
> get x:输出索引位置为x的元素的值,如果该位置没有元素或者索引不合法，输出-1
>
> getSize: 输出数组中的实际元素个数
>
> getCapacity:输出数组的容量

输入描述

第一行为正整数 N ， 表示有 N 条命令，下面 N 行为命令， 0 < N <= 50

示例1

```
输入:
5
add 0
add 1
remove 1
get 1
getSize
输出:
-1
1
```

示例2

```
输入:
15
add 0
add 1
add 2
add 3
add 4
add 5
add 6
add 7
add 8
```

```
add 9
get 1
add 10
get 10
getSize
getCapacity
输出：
1
10
11
15
解释：在add 10操作之后需要进行扩容
```

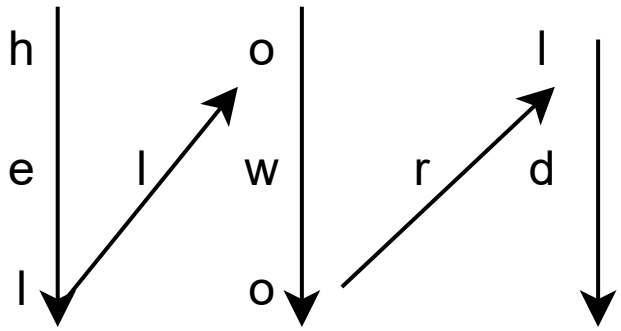
注意

不允许使用 STL 中的容器（包括但不限于 `string`、`vector` 和 `list`），一旦发现，**本题 0 分!**

Z形变换

将一个给定字符串根据给定的行数 N ，以从上往下、从左到右进行 Z 字形排列。

假设给定字符串"helloworld"， N取3，则按照如下箭头所示方向进行排列



输入描述

一个字符串，一个正整数 N ， N表示行数，空格隔开。字符串长度为[1,64]， 0 < N <= 20 ， **本题可用** `string`

输出描述

输出一个字符串，顺序从左到右，从上到下

示例

```
输入：PAYPALISHIRING 3
输出：PAHNAPLSIIGYIR
解释：Z形排列如下所示
P   A   H   N
A P L S I I G
Y   I   R
```

BF 解释器

Brainfuck 是一种极小化的程序语言，可以模拟图灵机进行工作。它基于一个简单的机器模型，除了指令，这个机器还包括：一个以字节为单位、元素全部被初始化为零并且大小无限的数组、一个指向该数组的指针（初始时指向数组的第一个字节，后文称为“数据指针”）以及用于输入输出的两个字节流。

基本知识

可将 8 个指令分为两类：数据操纵指令和控制流指令。

数据操纵指令定义如下：

指令	含义
`>`	数据指针向右移动一个位置
`<`	数据指针向左移动一个位置
`+`	数据指针指向的字节的值加一
`-`	数据指针指向的字节的值减一
`,`	从输入流中读取一个字节，将其存入数据指针指向的位置
`.`	将数据指针指向的字节写入输出流中

上述指令均为顺序执行。对于 `,` 指令，如果遇到 `EOF`，应当将数据指针所指的字节的值改为 `0`.

控制流包含两个：`[` 和 `]`，这类指令不会影响数据指针和数据指针指向的字节的值，只会影响下一条需要执行的指令。

指令	含义
`[`	如果数据指针指向的单元值为 `0`，向后跳转到对应的 `]` 指令的下一条指令处
`]`	如果数据指针指向的单元值不为 `0`，向前跳转到对应的 `[` 指令的下一条指令处

输入描述

输入包含若干行，第一行为 BF 程序，其余行为该 BF 程序所需要的输入（若有）。

保证 BF 程序一定是有意义、正确且能够结束的。保证模拟的数组长度不大于 1000.

输出描述

输出所有 `.` 命令的结果。

示例

示例 1

读取一个字符并输出。

输入

```
,.
a
```

输出

```
a
```

示例 2

打印“Hello”，五个字符的 ASCII 码分别是 `72` `101` `108` `108` `111`.

输入

```
+++++++++[->++++++++<]>.[[-]<]++++++++++[->++++++++++<]>+.[[-]<]+++++++++[-
>++++++++++++<]>.[[-]<]+++++++++[->++++++++++++<]>.[[-]<]+++++++++[->++++++++++++
<]>+++.
```

输出

```
Hello
```

注：末尾没有换行符

解释

将上述 BF 程序第一个输出命令 `.` 之前的命令取出分解为为如下

```
+++++++++
[->+++++++++<]
>.
```

在 `+++++++++` 之后数组变为如下状态：

9	**0**	**0**	**...**

`[->+++++++++<]` 的意思是：如果数据指针指向的字节的值不为 `0`，将该值减 1，然后将数据指针右移一个位置，将当前位置加 8，数据指针再左移一个位置。其实就是循环了 9 次，每次给数组第二个位置加 8，这段命令执行结束之后，数据指针指向第一个位置，数组如下：

0	**72**	**0**	**...**

然后 `>.` 将数据指针右移一个位置（在此之前数据指针指向数组的第一个位置），输出当前的内容，即 `72`，是字母 `H` 的 ASCII 码。

`[[-]<]` 这段命令的作用就相当于重置了第二个位置的内容为0，后面的逻辑类似。

提示

- 处理 `[` 和 `]` 命令不一定需要使用栈这种数据结构。
- 可使用如下方式读入 BF 程序（在 C++ 中，`char` 类型的变量可以用来表示一个字节，不过在输入以外的场景更推荐使用 `<cstdint>` 头文件中的 `uint8_t`）：

```
#include <iostream>
#include <string>

// 读取 BF 程序
string cmd;
std::getline(std::cin, cmd);
```

可参考如下方式处理 `,` 指令：

```
#include <cstdint>

char c;
uint8_t byte;
if (!std::cin.get(c)) {
    byte = 0;
}
```

螺旋打印矩阵

给定 m 行 n 列的矩阵，请顺时针打印矩阵的元素

输入描述

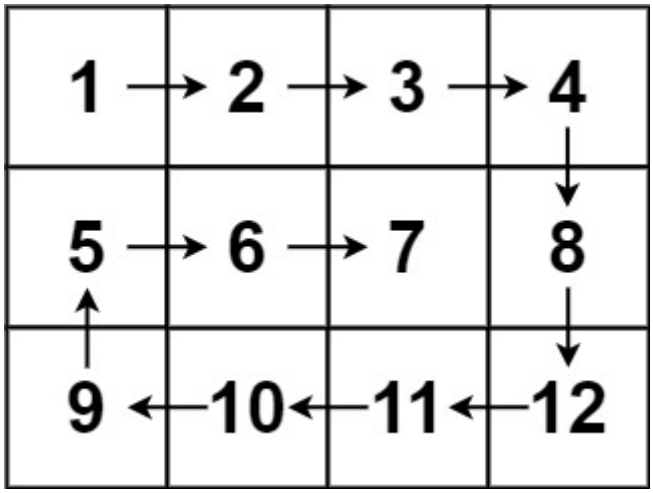
第一行为两个数，空格隔开，分别为 m 和 n 。接下来为 m 行 ， 每行 n 个元素，空格隔开。矩阵元素都是`int`类型。

其中 1 <= m,n <= 100

输出描述

元素之间以空格隔开。

示例




```
输入:3 4
    1 2 3 4
    5 6 7 8
    9 10 11 12
输出:1 2 3 4 8 12 11 10 9 5 6 7
```