Objective:

- Understanding and finding out the LIFO/FIFO behavior in different problems.

Task-1 Stack of Flapjacks

Background

Stacks and Queues are often considered the bread and butter of data structures and find use in architecture, parsing, operating systems, and discrete event simulation. Stacks are also important in the theory of formal languages.

This problem involves both butter and sustenance in the form of pancakes rather than bread in addition to a finicky server who flips pancakes according to a unique, but complete set of rules.

The Problem

Given a stack of pancakes, you are to write a program that indicates how the stack can be sorted so that the largest pancake is on the bottom and the smallest pancake is on the top. The size of a pancake is given by the pancake's diameter. All pancakes in a stack have different diameters.

Sorting a stack is done by a sequence of pancake ``flips''. A flip consists of inserting a spatula between two pancakes in a stack and flipping (reversing) the pancakes on the spatula (reversing the sub-stack). A flip is specified by giving the position of the pancake on the bottom of the sub-stack to be flipped (relative to the whole stack). The pancake on the bottom of the whole stack has position 1 and the pancake on the top of a stack of *n* pancakes has position *n*.

A stack is specified by giving the diameter of each pancake in the stack in the order in which the pancakes appear.

For example, consider the three stacks of pancakes below (in which pancake 8 is the top-most pancake of the left stack):

8	7	2
4	6	5
6	4	8
7	8	4
5	5	6
2	2	7

The stack on the left can be transformed to the stack in the middle via *flip(3)*. The middle stack can be transformed into the right stack via the command *flip(1)*.

The Input

The input consists of a sequence of stacks of pancakes. Each stack will consist of between 1 and 30 pancakes and each pancake will have an integer diameter between 1 and 100. The input is terminated by end-of-file. Each stack is given as a single line of input with the top pancake on a stack appearing first on a line, the bottom pancake appearing last, and all pancakes separated by a space.

The Output

For each stack of pancakes, the output should echo the original stack on one line, followed by some sequence of flips that results in the stack of pancakes being sorted so that the largest diameter pancake is on the bottom and the smallest on top. For each stack the sequence of flips should be terminated by a 0 (indicating no more flips necessary). Once a stack is sorted, no more flips should be made.

Sample Input

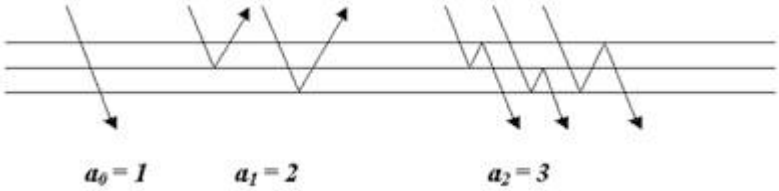
```
1 2 3 4 5
5 4 3 2 1
5 1 2 3 4
```

Sample Output

```
1 2 3 4 5
0
5 4 3 2 1
1 0
5 1 2 3 4
1 2 0
```

Task-2 *[Are you done with task-1? Then go for it.]*

Suppose we put two panes of glass back-to-back. How many ways a_n are there for light rays to pass through or be reflected after changing direction **n** times ? Following figure shows the situations when the value of **n** is 0, 1 and 2.



Input

It is a set of lines with an integer **n** where **0 <= n <= 1000** in each of them.

Output

For every one of these integers a line containing a_n as described above.

Sample Input

0
1
2

Sample Output

1
2
3

Task-3 ***[Are you done with task-2? Then go for it.]***

Roman Roulette

The historian Flavius Josephus relates how, in the Romano-Jewish conflict of 67 A.D., the Romans took the town of Jotapata which he was commanding. Escaping, Jospehus found himself trapped in a cave with 40 companions. The Romans discovered his whereabouts and invited him to surrender, but his companions refused to allow him to do so. He therefore suggested that they kill each other, one by one, the order to be decided by lot. Tradition has it that the means for affecting the lot was to stand in a circle, and, beginning at some point, count round, every third person being killed in turn. The sole survivor of this process was Josephus, who then surrendered to the Romans. Which begs the question: had Josephus previously practiced quietly with 41 stones in a dark corner, or had he calculated mathematically that he should adopt the 31st position in order to survive?

Having read an account of this gruesome event you become obsessed with the fear that you will find yourself in a similar situation at some time in the future. In order to prepare yourself for such an eventuality you decide to write a program to run on your hand-held PC which will determine the position that the counting process should start in order to ensure that you will be the sole survivor.

In particular, your program should be able to handle the following variation of the processes described by Josephus. $n > 0$ people are initially arranged in a circle, facing inwards, and numbered from 1 to n. The numbering from 1 to n proceeds consecutively in a clockwise direction. Your allocated number is 1. Starting with person number i, counting starts in a clockwise direction, until we get to person number k ($k > 0$), who is promptly killed. We then proceed to count a further k people in a clockwise direction, starting with the person immediately to the left of the victim. The person number k so selected has the job of burying the victim, and then returning to the position in the circle that the victim had previously occupied. Counting then proceeds from the person to his immediate left, with the kth person being killed, and so on, until only one person remains.

For example, when $n = 5$, and $k = 2$, and $i = 1$, the order of execution is 2, 5, 3, and 1. The survivor is 4.

Input and Output

Your program must read input lines containing values for n and k (in that order), and for each input line output the number of the person with which the counting should begin in order to ensure that you are the sole survivor. For example, in the above case the safe starting position is 3. Input will be terminated by a line containing values of 0 for n and k.

Your program may assume a maximum of 100 people taking part in this event.

Sample Input

```
1 1
1 5
0 0
```

Sample Output

```
1
1
```