Nested

For the purpose of this problem, we define a **container** to be an array-like data structure.

Suppose we have a **container** of *char*-type elements: '1', '2', '3', '4', '5', its string representation would look like the following:

```
<'1','2','3','4','5>
```

We access a **container**'s elements by indexing. **Containers** are 0-indexed.

However, the elements of a **container** may be **containers** themselves. We call this *nesting*. The nesting can go arbitrarily deep, and all sub-**containers** need not have the same length.

As an example:

```
myContainer = <'1',<'2'>,<'3','4',<'5','6'>>>
```

As you can see, *myContainer* above has 3 elements:

- myContainer[0] is a char-type '1'
- myContainer[1] is a container of length 1, whose sole element is char-type '2'
- myContainer[2] is container of length 2, whose first element is a char-type '3', second element is a char-type '4', and the third element is, again, a container of length 2. The innermost container contains 2 char-types, '4' and '5'.

You will be given a **container** in its string representation. Your task is to write a program that supports indexing into container. For example, for *myContainer* given above, an index-sequence "2 2 0" translates into myContainer[2][2][0], and should return '5'.

Keep reading for input and output specification.

Input Format

Each test case contains 2 lines. Line 1 is a string representation of a container. For Line 1, you may assume:

- the container is valid (no unmatched "<" or ">")
- no container or sub-container shall be empty
- the elements of the container are comma-separated
- there are no whitespaces
- the container's prmitive type is alway char-type, each char is enclosed by single-quotes ''
- A char can be [0-9A-Za-z!@#$%^&*()>

Line 2 represents a index-sequence, which is a series of space-delimited integers. As an example, the line

```
2 2 0
```

has the same effect of calling

```
thisContainer[2][2][0]
```

on the data structure whose string representation is given by Line 1

For Line 2, you may assume:

- no index will be out of bounds
- the index-sequence will always land on a char-type

Constraints

Constraints are described in "Input Format" section.

Output Format

For each testcase, output one character (without single quotes), which is the char that the index-sequence lands on.

Sample Input 0

```
<'1','2','3','4','5'>
3
```

Sample Output 0

```
4
```

Explanation 0

myContainer = <'1','2','3','4','5'> myContainer[3] --> '4' Notice the **4** is printed without quotes around it.

Sample Input 1

```
<'1',<'2'>,<'3','4',<'5','6'>>>
2 2 0
```

Sample Output 1

```
5
```

Explanation 1

myContainer = <'1',<'2'>,<'3','4',<'5','6'>>> myContainer[2][2][0] = '5' Notice the **5** is printed without quotes