1 Queue

An animal shelter has some dogs and cats, and **each animal has its own name (a unique string)**. When someone adopting an animal from the shelter, the adopter can choose to adopt a cat or a dog; but **adopter can only adopt the animal that enters the shelter first**. If there are no cats or dogs that the adopter want left, the adopter will decide to adopt the other animal. The shelter closed after all the animals were adopted. Please output the names of the adopted animals in order.

Input：

An string Array, length = n, represents the name of animal (in chronological order).

An string Array, length = n, represents the type(Cat/Dog) of animal (in chronological order).

An string array, length = m <= n, represents the type(Cat/Dog) of animal that the adopter wants.

Output:

An string array, length = m, represents the order in which the animals were adopted

Examples:

Input

`["A","B","C","D"]`

`["Cat","Dog","Dog","Dog"]`

`["Dog","Cat","Cat","Dog"]`

(explain: A is cat, B is dog, C is dog, D is Dog)

(first person want to adopt dog, so B is given; second want cat, A is given; third want cat, but there's no cat, C is given; ....)

Output

`["B","A","C","D"]`

Input

`["qwe","rty","uio","asd","fghj","kl","zxcvb"]`

`["Cat","Dog","Dog","Dog","Cat","Cat","Cat"]`

`["Cat","Dog","Cat","Dog"]`

Output

`["qwe","rty","fghj","uio"]`

```
public static String[] adoptOrder(String[] names, String[] type, String[] adoptType
    //use Queue to solve this problem is efficient!
}
```

2 Sort problem

Given an integer array `nums`, please sort the array in ascending order.

Use **Merge Sort** OR **Quick Sort**!!!

Input:

out-of-order integer array

Output:

integer array in ascending order

```
public static int[] merge(int[] array){
    //You shouldn't modify the array given! When you want to return an array, pleas
}
```

```
public static int[] quickSort(int[] array){
    //You shouldn't modify the array given! When you want to return an array, pleas
}
```

3 Stack Problem: computing the value of some balanced parenthesis

Meaning of the input string

There is a **string** `s`, it represents **some balanced parenthesis** that contains only left parentheses and right parentheses.

Balanced parenthesis means: all parenthesis in string can be matched in pair, such as: `()` , `(())` , `((()))()()` ;

and some examples of unbalanced parenthesis: `(` , `(()` , `)()(`

Valuation of String:

Each pair of parentheses values 1 point. **(`()` = 1)**

For any two arbitrary balance parenthesis strings S1 and S2, they have following relationships:

(S1)=S1*2 valuation of (()) = 1 * 2 = 2, valuation of ((())) = (1 * 2) * 2 = 4, ...

S1S2=S1+S2 valuation of ()() = 1 + 1 = 2, valuation of ()()() = 1 + 1 + 1 = 3, ...

What you need to do is to **calculate the valuation of the input string S**.

Input:

A string S: represents some balanced parenthesis. (Suppose all the inputs are balanced! But if you want to add a judge, it's OK)

Output:

Integer value: valuation of the input string.

Examples:

Input

`()`

Output

`1`

Input

`(())()`

Output

`3`

Input

`()(((())))`

Output

10

```
public static int score(String s) {
    //You should use Stack!
}
```