2003 Canadian Computing Competition, Stage 2 Day 1, Question 1

Input file: `bf.in`
Output file: `bf.out`
Source file: `n:\bf\bf.___`

BFed

For this problem, you will write an interpreter for an extremely simple programming language. The language does have a name, but for various, ahem, reasons, we shall call it *BF*.

A *BF* program operates on a simple 1-dimensional array of memory cells; there is a pointer to a "current" memory cell. In "vanilla" *BF*, this array has a size of 30,000, and each cell is an 8-bit integer - that is, cells store values in the range 0-255. Incrementing a cell with a value of 255 wraps around to 0; decrementing a cell with a value of 0 wraps around to 255. All cells are initially set to 0, and the pointer initially points to the leftmost cell.

The *BF* program consists of a string. Each character can be one of 8 "commands":

>	move the pointer right one cell
<	move the pointer left one cell
+	increment the current cell by 1
−	decrement the current cell by 1
[	skips ahead to the matching] IF the current cell contains 0
]	returns to the matching [UNLESS the current cell contains 0
.	outputs the current cell as a character
,	inputs one character into the current cell

You do not need to implement the ',' command.

You should ignore any characters in the *BF* program except the first 7 commands listed above. The program ends when there are no more characters to be processed.

Interestingly enough, these commands are powerful enough that a *BF* program can (given sufficient memory, time, and programming patience) perform any computation that any other programming language can do!

Input

Your interpreter will be given a *BF* program in standard input. It may span multiple lines. The program will be terminated by a hash mark (#).

You may assume that no programs will be given to your interpreter that are invalid, run unreasonably long (or forever), or crash off the left or right end of the array. No input will be longer than 10,000 characters.

Output

Your interpreter should print the output from the execution of the *BF* program. Do not print any characters other than the ones from the program.

Sample Input 1

```
++[>+++++++++++++<-]             // put 26 in cell 1
>>>++++++++[<++++++++>-]<+       // put 65 in cell 2
<[->.+<]                         // output alphabet
++++++++++.                      // output newline
#
```

Sample Output 1

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Sample Input 2

```
[+[>uh-oh<]+]------outer[+>------inner[+>-----<]<]>>.<++++++++++.#
```

Sample Output 2

```
L
```

Sample Input 3

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

Sample Output 3

```
Hello World!
```

2003 Canadian Computing Competition, Stage 2
Day 1, Question 2

Input file: `cards.in`
Output file: `cards.out`
Source file: `n:\cards\cards.___`

Concentration

Stan has a deck of N Concentration Cards. He wants to lay the cards edge-to-edge to form a filled rectangle with minimal perimeter. Each card is a rectangle with dimensions W mm. by H mm.

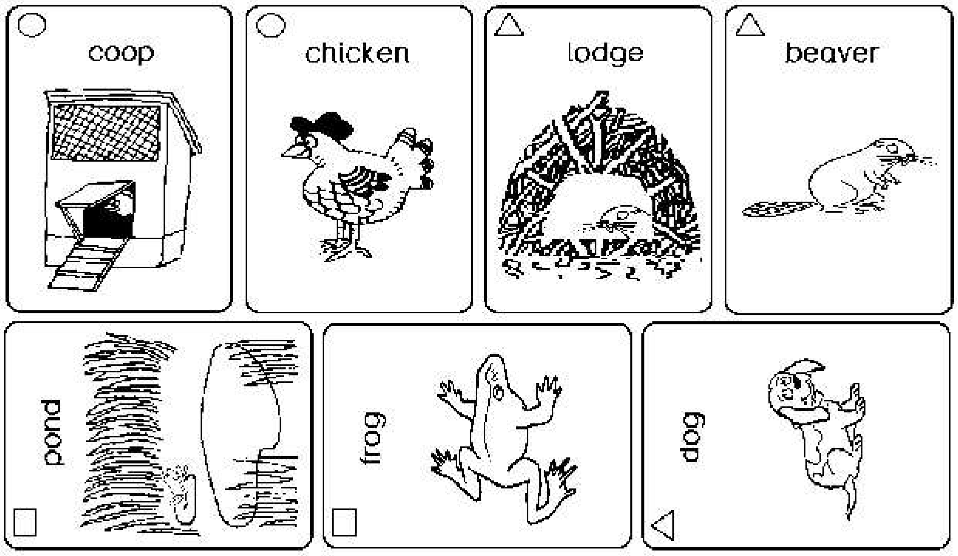

Figure 1: Concentration Cards

Input
The first line of input contains C, the number of test cases. For each case there is an additional line containing N, W, H, each a positive integer not exceeding 1000.

Output
Your program should produce one line of output per case, giving the minimal perimeter.

Sample Input

```
3
3 300 400
4 400 300
7 300 400
```

Sample Output

```
2600
2800
3800
```

2003 Canadian Computing Competition, Stage 2
Day 1, Question 3

Input file: `cube.in`
Output file: `cube.out`
Source file: `n:\cube\cube.___`

Cube

Imagine a cube formed from solid interlocking pieces of various shapes. If the pieces are sufficiently entwined, the only way to separate them would be to cut some of them. We can ask the question: "is the cube stable?" That is, is it physically impossible to separate the cube into 2 or more fragments without deforming or cutting any individual piece?

Your program must answer this question for a variety of such cubes.

The pieces that make up a cube will be specified as follows: divide the cube into a grid of $n * n * n$ miniature cubes, each labeled by a capital letter. Two adjacent (face-sharing) minicubes are joined together if and only if they are labeled by the same letter. For instance, the first example cube given consists of 3 solid pieces.

Input

Your program will be given the specification of up to 10 different cubes. The first two lines of each specification will consist of the size of that cube, n ($1 \leq n \leq 10$), and a blank line. The remaining $n * (n + 1)$ lines will specify the n horizontal layers of the cube from bottom to top. Each layer specification consists of an $n * n$ square showing the labels for each minicube on that layer, followed by a blank line. There will be no spaces in the input. The input will be terminated by the number 0 on a line by itself.

Output

For each cube given, in the order specified, print "Yes" if that cube is stable, and "No" if it is not.

Sample Input

```
2

AB
AB

BB
BA

3

AAA
BBB
AAA

AAA
ABA
AAA

ABA
ABA
ABA

0
```

Sample Output

```
No
Yes
```