Problem A : Best Fishing Pattern

A program is needed to find the best fishing pattern. A pattern consists of how deep a lure is presented followed by the lure colors. Depth is indicated by a single word with no whitespace as are colors. For example, *deep yellow blue* means that the lure is fished deep and has the colors yellow and blue. There can be two or three colors. The purpose of the program is to input an unknown number of patterns, one per line, and output the pattern that occurs most often.

There is at least one fishing pattern input. Each input line contains a depth followed two or three colors. A single space is used to separate words. All words are lowercase; thus, case sensitivity is not an issue—there is no need to check for Blue and blue being the same, for example. Colors can be entered in any order; thus, *red yellow green* is the same color combination as *yellow green red*. It is guaranteed that there is one best fishing pattern.

The best pattern is to be output. Because colors for a pattern can be entered in any order, the output pattern must be the first occurrence of the best pattern in the input.

Input

```
deep red white
surface blue white yellow
surface blue white yellow
shallow blue silver red
medium blue white
shallow silver red blue
surface gold green
shallow silver red blue
```

Output

```
shallow blue silver red
```

End of Problem A

Problem B : Haul Visual

A program is needed to graphically show the haul of fishing boats relative to the average, or mean, of all hauls. Input consists of one or more integers, each in the range 0 to 50. There is one integer per line.

The mean is an integer equal to truncating the actual float result. For example, 17.93 is truncated to 17, and 26.2343 is truncated to 26.

The output is a graph showing the hauls relative to the mean. Smaller hauls are to the left, and larger ones are to the right. The vertical axis represents the mean.

Input

```
23
29
7
22
15
20
26
```

Output

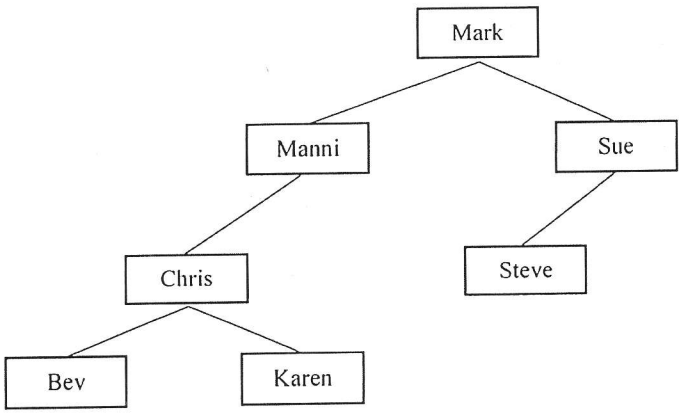
```
Mean 20
                  |+++ 23
                  |+++++++++ 29
7 ----------------|
                  |++ 22
         15 ------|
                  | 20
                  |++++++ 26
```

End of Problem B

recursion

Problem C : Binary Search Tree

A program is needed to calculate the average (mean) depth of all nodes in a binary search tree. For example, if the inputs are Mark, Manni, Sue, Chris, Bev, Karen, and Steve (in this order), then the resulting binary search tree is:



Mark's depth (the root) is 1, Manni 2, Sue 2, Chris 3, and so on. The mean depth is then $(1 + 2 + 2 + 3 + 3 + 4 + 4) / 7 \approx 2.7143$.

In a binary search tree, each name is placed in alphabetical order relative to names that are already in the tree. The first input node is the root. Each remaining name is then placed as a leaf in the current tree when the name is input. This placement is to the left if the new name is less than a name already in the tree and to the right if the new name is greater.

Input is one name per line. Names contain no whitespace and are case sensitive. Mark and mark, then, are different names. The only output is the mean depth rounded to four decimal places.

Input

```
Mark
Manni
Sue
Chris
Bev
Karen
Steve
```

Output

```
2.7143
```

End of Problem C

Problem D : Grammar

A program is needed to check that sentences match their grammar rules. Each input line contains a grammar rule followed by a space followed by a sentence. The program is to indicate, Yes or No, whether each sentence matches its grammar. Neither grammar rules nor sentences contain whitespace.

All grammars consist of only alphabetic characters. The alphabetic character in the string indicates a single occurrence of that character in the grammar unless it is followed by *, which means zero or more, or +, which means one or more. A digit 2-9 means that a symbol occurs exactly as many times as the digit indicates.

Input

```
a2d*b+ aabbbb
ab*c*d3 acdd
b*gf2 bbbbbff
p+q2a3 pqqaaa
```

+ - one or more

2-9 - exactly that many

* - none or more

Output

```
Yes
No
No
Yes
```

End of Problem D

strPos

a2a3

nothing

word

b+c+d+

3 2 X + 2 3 2 1

1 3 1

2 1 2 1

2 2

1 3 2 4 1 2 3

Problem E : Ball In/Out of Bounds

A program is needed to count the number of times a ball is in and out of bounds. The bounds is indicated by a rectangle—the playing zone. A ball is considered in-bounds if it is inside the rectangle or on its border. There are 1 or more ball locations, each indicated by a single digit 1-9. A 1 means that one time a ball hit the playing zone at that location. A 2 means that twice a ball hit the zone at the location. And, so on. X's indicate the playing zone.

The first input line shows either one or more locations where the ball is out of bounds before the playing zone or the first (top) bounds of the playing zone, possibly with ball locations. Input lines can be empty. This happens when there are no ball locations on a line outside the playing zone. Input lines contain only spaces, the digits 1-9, and X. The entire input is no more than 50 lines by 80 characters—space, 1-9, or X—per line.

Output is the number of times a ball is in and out of bounds, each preceded by either In or Out and a space.

Input

```
   1    1
 1 XXXX1XXX
   X2     X
   X      X
   X    3 X  1
   X      1
   XXXX1XXX

         2
```

Output

```
In 8
Out 6
```

End of Problem E

Problem F : Drone Flight

A program is needed to show, in a simple textual way, the flight of a drone. The drone always moves forward (to the right). Input to the program is a sequence of hexadecimal digits describing the flight. Each binary digit in a hexadecimal digit describes how the drone moves: 1 means up and 0 means down. The drone does not fly level.

The drone always starts from the ground. If a drone tries to fly down but is already on the ground (either at the start or later), then the drone simply stays on the ground. This no-change, which can only occur when the drone is on the ground, is shown in the flight as one or more successive positions of the drone on the ground.

The maximum height that the drone can fly, once off of the ground, is fifty. The output, then, can have at most fifty-two rows:

- The top fifty rows, as needed, show the drone (D) in flight.
- The next row shows the drone on the ground.
- The last row shows the ground (X).

The ground is shown to the exact end of the drone's flight (the last X aligns in the same column as the last—rightmost—position of the drone).

There is at least one and no more than ten hexadecimal digits in the input. The first column in the output depicts the starting point of the drone on the ground. The remaining output shows the drone's flight according to the input. Note that there is no "partial" last hexadecimal digit in the input—the flight is always recorded in full hexadecimal digits.

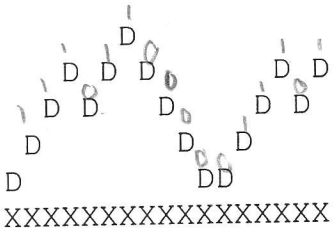
The output shows only the heights that the drone reaches—any heights (rows) not reached by the drone are not shown.

E = 1110
C = 1100
1 = 0001
D = 1101

Input

```
EC1D
```

Output



End of Problem F

Problem G : Truth Table

A program is needed to output the truth tables for one or more Boolean expressions. Input is one or more lines, each having the form *variable op variable* [*op variable*]*. *variable* is any of the twenty-six uppercase characters A-Z. *op* is one of the five operators **and**, **or**, **xor**, **nand**, or **nor**. Operators are lowercase only.

Each input expression, at a minimum, has a variable followed by an operator followed by a variable. There can be zero or up to four additional operators, each followed by a variable. There is no operator precedence—all operators are evaluated left to right. Parentheses are not permitted in an expression. The same variable can occur more than once in the same expression. A single space separates variables and operators.

Output is the truth table for each input expression. Variables in the truth table must be in the same order as they first occur in the expression. Furthermore, the output truth table is to be perfectly neat. Successive truth tables are separated by a blank line.

Input

```
A or D and P nand D
A xor B
```

Output

```
A D P Output
0 0 0 1
0 0 1 1
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 1
1 1 0 1
1 1 1 0

A B Output
0 0 0
0 1 1
1 0 1
1 1 0
```

End of Problem G