CS100 Homework 3 (Spring, 2022)

Deadline: 2022-03-17 00:02:00

Late submission will open for 24 hours after the deadline, with -50% point deduction.

Problem 1. God's Language

Liu Big God, who loves pure math, has read many books about algebraic geometry recently. Today he found that human languages are very interesting. To his surprise, some words or sentences read the same backward or forward, like "racecar" and "Was it a car or a cat I saw?".

We define a string to be a **palindrome** if its reverse is the same as itself. Now Liu Big God is curious about whether a given string is a palindrome. Since Liu Big God loves pure math very much, he does not care about the digits or punctuations. Also, he treats upper- and lower-case letters as the same. In short, when judging whether a string is a palindrome, you only need to consider the English letters in it, and treat lower- and upper case letters as the same. For example, the following line is a palindrome:

`a3423bcd.,/ 351DC ba.+=`

In this string, after removing the whitespaces, punctuations and digits, it becomes `abcdDCba`. In a case-insensitive way, this is regarded as a palindrome.

Considering that Liu Big God is a god, he has the right to do some modification to the string. Initially he has a string S, in which the characters are indexed $0, 1, 2, \cdots$. Liu Big God will send you m commands, each of which being one of the following:

- `0` l r, where l and r are nonnegative integers indicating two indices in S. You need to check whether the substring $S[l, \cdots, r]$ is a palindrome according to the rules above. Output `Yes` if it is a palindrome, and `No` otherwise. It is guaranteed that $0 \leqslant l \leqslant r < \text{size}(S)$, where $\text{size}(S)$ is the number of characters in the string S.
- `1` k c, where k is a nonnegative integer and c is a character. You need to replace the character indexed k in S with the character c. It is guaranteed that $0 \leqslant k < \text{size}(S)$. Notice that there is only one space between `1` and k.
- `2` s, where s is a string that begins right after the space after '`2`' and ends right before the newline character. You need to append s to the end of S.

It is guaranteed that $1 \leqslant m \leqslant 5000$, and that the length of the string will not exceed 5000 at any time.

Sample input

```
No lemon, no melon!
8
0 6 12
0 10 17
1 2 x
2 INOL
0 15 22
2  35./e100m
0 5 5
0 13 32
```

Sample output

```
Yes
No
Yes
Yes
Yes
```

Problem 2. Code formatter

Writing code is like writing poetry. Everyone has their style, but the bottom line is that people should enjoy reading it. Good code can be read and understood in a short time with minimal thinking. Keeping a good coding style will be very beneficial to your future study. You may have a glance at Google C++ Coding Style for details.

A well-organized code should at least look neat. An example is shown as follows.

```c
#include <stdio.h>
int main(){
for(int i=1;i <=9;i++)
{
for(int j=1;j<= i; j++)
printf("%d*%d=%d\t",j,i,i*j);printf("\n");}
return 0;
}
```

```c
#include <stdio.h>

int main(){
    for(int i = 1; i <= 9; i++) {
        for(int j = 1; j <= i; j++) {
            printf("%d*%d=%d\t", j, i, i*j);
        }
        printf("\n");
    }
    return 0;
}
```

Figure 1: **Left**: Bad example **Right**: Good example

Although both of the codes above are runnable, you may feel uncomfortable with the code on the left. The differences are the **indentation**, **spaces**, and **lines**. When you are coding something, try to create a good measure of the vertical density of your code and keep it consistent.

In this problem you are going to format a code snippet according to the following rules:

Indentation

Indent your code so that it corresponds to the logical structure of the program. (such as `for` and `while` loop and `if` condition) You should use only spaces for indentation, and indent 4 spaces at a time.

An example is shown as follows:

```
/* Before indentation*/
if(a > b) {
      printf("The larger one is\n");
printf("a\n");
}
/* After indentation*/
if(a > b) {
    printf("The larger one is\n");
    printf("a\n");
}
```

Curly Braces

There are many indentation styles for brace placement. Here we follow Stroustrup's style. The open curly brace is always at the end of the last line of its respective control statements. The close curly brace

is always on the last line by itself. There should be **a space** between the close parenthesis and the open curly brace. Similarly, there should be **a space** between the keyword `else` and the open curly brace.

```
/* For loop */
for(int i = 0; i < 5; i++) {
    printf("%d ", i);
}
/* If-else */
if(x < 0) {
    puts("Negative");
}
else {
    puts("Non-negative");
}
```

Line

A simple C statement should always end with a semicolon ';'. If a line has multiple statements, you need to put them into multiple lines one by one. For example,

```
/* Before */
statement1; statement2; statement3;
/* After */
statement1;
statement2;
statement3;
```

Please make sure to add a new line `'\n'` **right after** the semicolon `';'` and remove the redundant spaces.

Input description

You will receive a code snippet in C which is not well-formatted. You need to output a formatted version of the code snippet by the rules above. You only need to consider improving the indentation, curly braces, and lines of the code. So the logic and contents of the code are not useful to you.

You will receive an integer n in the first line indicating the line of code. The following n lines are the code snippet.

We have made some simplifications of the code snippet:

- There are no bugs in the code that may lead to compile errors. In other words, all the parenthesis and curly brace will be matched. There are no missing semicolons.
- Only the following control flow statement will be included: `if`, `else if`, `else`, `for`, `while` and their combination.

- No statement will be split into several lines.
- There are no blank lines and comments in the code.
- No code will be embedded in the strings.
- The open curly brace is always on the same line of its respective control statements. The close curly brace is always on the last line by itself. **But the indentation of the close curly brace and the placement of the open curly brace is arbitrary**.
- No control statements will be at the same line. For instance, `for` statement and `if` will not be at the same line.
- Each line has at most 80 characters.

In summary, your code formatter should make sure

- All the indentations are well-organized.
- Each line has at most one statement.
- The placement of braces follows Stroustrup's style.

Sample input 1

```
4
int a = 5;
if(a > 1){
a = 1; printf("done\n");
}
```

Sample output 1

```
int a = 5;
if(a > 1) {
    a = 1;
    printf("done\n");
}
```

Sample input 2

```
10
for(int i = 1; i <= 9; i++){
for(int j = 1; j <= 9; j++) {
if(j <= i){
printf("%d*%d=%d\t", j, i, i*j);       printf("\n");
}
else{
```

```
8   continue;
9   }
10  }
11  }
```

Sample output 2

```
1   for(int i = 1; i <= 9; i++) {
2       for(int j = 1; j <= 9; j++) {
3           if(j <= i) {
4               printf("%d*%d=%d\t", j, i, i*j);
5               printf("\n");
6           }
7           else {
8               continue;
9           }
10      }
11  }
```

Problem 3. Figure Skating

In Beijing Winter Olympics, the highly noted Japanese figure skater Yuzuru Hanyu challenged a 4A jump (the hardest single jump), being the first one to perform a 4A jump in the Winter Olympics history. While figure skating is of great aesthetics to watch, the judging process of figure skating is extremely complicated. Not only should the judge system comprehensively consider both the technical difficulty and the execution of skaters' performance, but the scale judging panel is also immense (consisting of more than 9 judges and 5 technical specialists).

Score of each element in a figure skating performance is based on both a base value, which measures the technical difficulty of each move, and a grade of execution(GOE), measuring how well the skater has performed. According to Figure skating - Wikipedia and this Scoring Cheat Sheet, here are some simplified rules to calculate the base values for jumps in figure skating:

Jump Codes and Scores

A jump is recorded in form of a code (e.g. 4A, 3Lz), which includes a number, indicating the number of rotations the skater has performed in mid-air, and an abbreviation of the jump type. There are six different types of jumps. With different rotations, they are scored by the following table:

	1	2	3	4
T	0.4	1.3	4.2	9.5
S	0.4	1.3	4.3	9.7
Lo	0.5	1.7	4.9	10.5
F	0.5	1.8	5.3	11.0
Lz	0.6	2.1	5.9	11.5
A	1.1	3.3	8.0	12.5

This table, along with a function `double Score(int rotations, enum JumpType type)`, has been provided in the code templates. You can easily look up the scores by writing:

```
1 double score = Score(4, A); // score = 12.5
```

Error Annotations

A skater's performance may contain errors, for example, spinning less rotations than intended, or taking off in a wrong pose. In such cases, jump codes will be annotated with marks. (e.g. 3A<, 3Lo<<) They influence the base values by the following rules:

q	does not affect the base value
<	reduces the base value to 70% of its original value
<<	evaluates as the same jump with one less rotation

There are also errors specifically for **F** and **Lz** jumps:

!	does not affect the base value
e	reduces the base value to 70% of its original value

- A jump may only receive one error mark per category (one per chart above). For example, **2Tq<** is invalid.
- For an **F** or **Lz** jump, the error marks '!' and 'e' are written to the left of 'q', '<', and '<<', and their effects stack multiplicatively. For example, a **3Fe<** will receive a final base value of $5.3 \times 0.7 \times 0.7 = 2.597 = 2.60$, and a **2Lze<<** will have $0.6(1\text{Lz}) \times 0.7 = 0.42$.
- If a jump of 1 rotation is marked '<<', its score will be 0.00, but the jump is still valid.

Sequenced Jumps

A skater may perform up to 3 jumps in a sequence. Jump codes in a sequence are connected by '+' signs, and their base values are added together to calculate the total base value of the sequence. If the sequence is performed in the second half of a performance, it will be annotated with an **'x'** in the end, and the final base value of this sequence will be multiplied by 1.1.

- For example, a sequence **3Lze+3Tx** has a final base value of $(5.9 \times 0.7 + 4.2) \times 1.1 = 9.163 = 9.16$.

Now you are ready to become a technical specialist! Other judges are sending you jump sequences, and your job is to calculate base values for them. Some judges are so sloppy that the sequences they give you are **invalid**. For example:

- 3B, 3lZe (wrong types, and we are case-sensitive)
- 0A, 5F<<(wrong numbers of rotations)
- 3Lo?+2T, 4Ae, 3T+2Txx (wrong annotations)

If a sequence contains **any** invalid jumps, the base value for **the whole sequence** becomes 0.00.

Input description

You will receive 3 lines of input, each line containing a jump sequence. The sequences may not be valid.

Output description

You need to provide 3 lines of output, one for each line of input, containing the base value of the sequence, formatted to 2 decimal digits.

It's **both OK** whether you calculate and output after receiving each line of input, or you read all input lines and then calculate. In other words, it does not matter whether you have 3 lines of output together or in between the input lines when you test on your own machine.

Sample input

```
0A+1T
3Fe<
3Lze+3Tx
```

Sample output

```
0.00
2.60
9.16
```