# Problem 3. Calculator

## Description

Class Exercise 2

Implements a basic calculator. The program reads two numbers and an operator from the user and then performs a mathematical operation based on the provided operator. The supported operators are `'+'`, `'-'`, `'*'`, and `'/'`. The program then prints the result of the operation. This is an example in the class.

Then modify the program to make it run repeatedly in a loop, allowing the user to perform multiple calculations without having to restart the program and asking the user whether he wants to continue to use the calculator or quit the calculator.

## Input format

- Each line contains a real number $x_i$, an character $op_i$ and a real number $y_i$, separated by a space. The following line contains a character `y` or `n`.
  - For 100% cases, $|x_i| \leq 100$ and $|y_i| \leq 100$. Division by zero won't happen.
- The program stops when `n` is read from the input.

## Output format

- For each calculation, if $op_i$ is a supported operator, output the result in the format `xxx p yyy == zzz`, where `xxx` and `yyy` are replaced with $x_i$ and $y_i$ respectively, `p` is replaced with the operator $op_i$, and `zzz` is replaced with the calculation result. Otherwise, output `Unknown operator!` with a newline (`'\n'`) in the end. Then, print `Do you want to continue? (y/n)` on the next line, with a newline in the end.
- Print the floating-point value directly, without caring about the number of decimal places it should be rounded to.

## Example

```
1 + 2.5
1.000000 + 2.500000 == 3.500000
Do you want to continue? (y/n)
y
1 - 2
1.000000 - 2.000000 == -1.000000
Do you want to continue? (y/n)
y
1 & 2
Unknown operator!
Do you want to continue? (y/n)
n
```

Explanation: The input is as follows

```
1 + 2.5
y
1 - 2
y
1 & 2
n
```

The output is

```
1.000000 + 2.500000 == 3.500000
Do you want to continue? (y/n)
1.000000 - 2.000000 == -1.000000
Do you want to continue? (y/n)
Unknown operator!
Do you want to continue? (y/n)
```

## Notes

You don't have to print things after input is completely done.

This problem should be solved without arrays or dynamic memory allocation. Keep your solution simple.

Does `scanf("%c", ...)` skip the leading whitespaces in the input? If you want to read a `char` in this way, make sure you understand its behavior fully.