5803 Decoding EDSAC Data

The world's first full-scale, stored-program, electronic, digital computer was the EDSAC (**E**lectronic **D**elay **S**torage **A**utomatic **C**alculator). The EDSAC had an accumulator-based instruction set, operating on 17-bit words (and 35-bit double words), and used a 5-bit teletypewriter code for input and output.

The EDSAC was programmed using a very simple assembly language: a single letter opcode followed by an unsigned decimal address, followed by the the letter 'F' (for *full word*) or 'D' (for *double word*). For example, the instruction "`A 128 F`" would mean "*add the full word at location 128 to the accumulator*", and would be assembled into the 17-bit binary value, `11100000100000000`, consisting of a 5-bit opcode (`11100` = "*add*"), an 11-bit operand (`00010000000` = 128), and a single '`0`' bit denoting a *full word* operation (a 1 bit would indicate a *double word* operation).

Although arithmetic on the EDSAC was *fixed point two's complement binary*, it was not mere integer arithmetic (as is common with modern machines). The EDSAC hardware assumed a *binary point* between the leftmost bit and its immediate successor. Thus the hardware could handle only values in the range $-1.0 \leq x < 1.0$. For example:

Value	Binary Representation
-1.0	10000000000000000
1/2	01000000000000000
3/4	01100000000000000
-1/2	11000000000000000

As you can see, the largest possible positive value was:

$$01111111111111111 = 0.9999847412109375$$

and the smallest possible positive value was:

$$00000000000000001 = 2^{-16} = 0.0000152587890625$$

(This also happens to be the increment between successive values on the EDSAC).

By a curious coincidence (or an elegant design decision), the opcode for the *add* operation (`11100`) was the same as the teleprinter code for the letter '`A`'. The opcode for *subtract* was the same as the teleprinter code for '`S`' (`01100`), and so on. This simplified the programming for the assembler (which, incidentally, was a mere 31 instructions long). The EDSAC teleprinter alphabet was "`PQWERTYUIOJ#SZK*?F@D!HNM&LXGABCV`" (with '`P`' = `00000`, '`Q`' = `00001`, and so on, up to '`V`' = `11111`).

Unfortunately, the EDSAC assembler had no special directives for data values. On the other hand, there was no reason that ordinary instructions couldn't be used for this, thus, an EDSAC programmer desiring to reserve space for the constant 3/4 (represented as 01100000000000000) would use the instruction '`S 0 F`' and for 1/3 (which is approximately represented as 00101010101010101) '`T 682 D`', and so on.

Your job is to write a program that will translate EDSAC instructions into the appropriate decimal fractions.

Input

The first line of input contains a single integer P, ($1 \leq P \leq 1000$), which is the number of data sets that follow. Each data set is a single line that contains N (the dataset number), followed by a space,

followed by an EDSAC instruction of the form: '$c \sqcup d \sqcup s$', where c is a single character in the EDSAC alphabet, d is an unsigned decimal number ($0 \leq d < 2^{11}$), and s is either a '`D`' or '`F`'.

Note: $\sqcup$ represents a single space.

Output

For each data set there is one line of output. It contains the data set number (N) followed by a single space, followed by the exact decimal fraction represented by the by the EDSAC instruction, including a minus sign (for negative values). The format for the decimal fraction is: $sb.ddd\ldots$, where s is an optional minus sign, b is either a '`1`' or '`0`', and d is any decimal digit (`0-9`). There must be at least 1 and at most 16 digits after the decimal point. Trailing zeros in the fraction **must** be suppressed.

Sample Input

```
13
1 P 0 F
2 I 0 F
3 & 0 F
4 ? 0 F
5 Q 1228 D
6 P 0 D
7 V 2047 D
8 * 2047 D
9 ? 0 D
10 P 256 F
11 V 1536 F
12 T 682 D
13 T 54 F
```

Sample Output

```
1 0.0
2 0.5
3 -0.5
4 -1.0
5 0.0999908447265625
6 0.0000152587890625
7 -0.0000152587890625
8 0.9999847412109375
9 -0.9999847412109375
10 0.0078125
11 -0.015625
12 0.3333282470703125
13 0.31414794921875
```