ACM-style Programming Contest
Team Selection Trials
Contest One — Sept. 28, 1996

Rules:

1. All questions requre you to read the test data from standard input and write results to standard output.
   - No whitespace should appear at the end of a line, and all lines should be terminated with a new-line.
   - Tabs should never be used.
   - Output must correspond *exactly* to the provided sample output, including spelling and spacing. Multiple spaces will not be used in any of the judges output, except where expressly stated.
2. All programs will be re-compiled prior to testing with the judges' data. For this reason, non-standard libraries should not be used in your solutions.
3. Programming style is not considered in this contest. You are free to code in whatever style you prefer.
4. All communication with the judges will be handled by the submit command. If you have need of books, ask the helpers in the room.
5. Judges' decisions are to be considered final. No cheating will be tolerated.
6. There are five questions to be completed in three hours.

Question 1 — Bean Counting

The CSC has decided to hold a contest to guess the number of beans in a given jar. To make it a wee bit more interesting, they decided to have the contestants guess the number of a particular kind of bean, the jar having many types of beans in it. Each participant would pay $2.00 per guess.

If the guess is one off (either one greater or less) than the actual count, they get a single share of the total pot. If the guess is right on, they get two shares of the total pot.

Your task is to tally these guesses and print out how much each person wins.

Input:

The first line will containing hundreds of letters, from a–z, each representing one bean of that type of bean. There will only be 26 different types of beans. Each following line will be the guesses from a contestant. A contestant can guess as many times as they like. The line will begin with a single letter, in the range A–Z, leaving only 26 possible contestants. The first letter will be followed by a single space, and then up to five pairs of letter/number combinations, the letter representing the type of bean for this guess, and the number the number of that bean the contestant expects to find. The letter and the value of a guess will be separated by a ':' and each guess on lines with more than one guess will be separated by a ','.

Tallying:

The amount a contestant will receive is based on shares of a pot. The pot will consist of the total of all the guesses, at $2.00 per guess. The fraction each person gets will be based on:

TotNum	total number of guesses by all contestants
TotShares	total number of shares from all contestants
OffOne	number of off-by-one guesses by that contestant
RightOn	number of right-on guesses by that contestant

$$\frac{TotNum * \$2.00}{TotShares} * (OffOne + (RightOn * 2))$$

Output:

Print out all the winners, in alphabetical order, and their winnings, separated by a single space. Each winner should be on a line by itself.

Sample Input:

```
aaddddddddddddddddddddddddffffffffffffffffwwwwwwwwwwwwccccalalala
A a:6,c:6
B w:12
C d:25
D d:22,c:4,l:3,f:15
```

Sample Output:

```
A 3.20
B 3.20
D 9.60
```

Question 2 — Factoring Large Numbers

One of the central idea behind much cryptography is that factoring large numbers is computationally intensive. In this context one might use a 100 digit number that was a product of two 50 digit prime numbers. Even with the fastest projected computers this factorization will take hundreds of years.

You don't have those computers available, but if you are clever you can still factor fairly large numbers.

The Input:

The input will be a sequence of integer values, one per line, terminated by a negative number. The numbers will fit in gcc's `long long int` datatype, however `scanf` and `printf` do not appear to handle this datatype correctly, so you must do your own I/O conversion.

The Output:

Each positive number from the input must be factored and all factors (other than 1) printed out. The factors must be printed in ascending order with 4 leading spaces preceding a left justified number, and followed by a single blank line.

Sample Input:

```
90
1234567891
18991325453139
12745267386521023
-1
```

Sample Output:

```
    2
    3
    3
    5

    1234567891

    3
    3
    13
    179
    271
    1381
    2423

    30971
    411522630413
```

Question 3 — Maze Traversal

A common problem in artificial intelligence is negotiation of a maze. A maze has corridors and walls. The robot can proceed along corridors, but cannot go through walls.

<u>The Input:</u>

For this problem, you will input the dimensions of a maze, as two integer values on the first line. Of the two numbers, the first is the number of rows and the second is the number of columns. Neither the number of rows nor columns will exceed 60.

Following these numbers will be a number of rows, as specified previously. In each row there will be a character for each column, with the row terminated by the end of line. Blank spaces are corridors, asterisks are walls. There needn't be any exits from the maze.

Following the maze, will be an initial row and column specified as two integers on one line. This gives the initial position of the robot. Initially the robot will be facing North (toward the first row).

The remaining input will consist of commands to the robot, with any amount of interspersed white-space. The valid commands are:

R rotate the robot 90 degrees clockwise (to the right)

L rotate the robot 90 degrees counter-clockwise (to the left)

F move the robot forward, unless a wall prevents this, in which case do nothing

Q quit the program, printing out the current robot row, column and orientation

<u>The Output:</u>

The final row and column must be integers separated by a space. The orientation must be one of N,W,S,E and separated from the column by a space.

<u>Sample Input:</u>

```
7 8
********
* * * **
* *    *
* * ** *
* * *  *
*   * **
********
2 4
RRFLFF FFR
FF
RFFQ
```

<u>Sample Output:</u>

```
5 6 W
```

Question 4 — Compound Words

You are to find all the two-word compound words in a dictionary. A two-word compound word is a word in the dictionary that is the concatenation of exactly two other words in the dictionary.

The Input:

Standard input consists of a number of lowercase words, one per line, in alphabetical order. There will be no more than 120,000 words.

The Output:

Your output should contain all the compound words, one per line, in alphabetical order.

Sample Input:

```
a
alien
born
less
lien
never
nevertheless
new
newborn
the
zebra
```

Sample Output:

```
alien
newborn
```

Question 5 — Pit Stop Strategy

The speed of a racing car, all other factors being equal, depends on the amount of fuel it carries. In general, the weight of the fuel slows the car. In addition, the weight of the fuel increases fuel consumption. It is therefore an advantage to carry as little fuel as possible. Any amount of fuel may be loaded at the beginning of the race, and refuelling pit stops may be made during the race. The time consumed for the pit stop increases with the amount of fuel loaded. Your task is to determine the optimal fueling and pit stop strategy for each of a number of cars, based on measurements taken immediately before the race.

The Input:

Standard input consists of several lines of input, each containing:

- the number of laps in the race (integer less than or equal to 100)
- the theoretical lap time [seconds] of the car on an empty tank (float)
- the increase in lap time (float) per litre of fuel carried at the beginning of the lap (float)
- the theoretical fuel consumption [litres per lap] on an empty tank (float)
- the increase in fuel consumption per litre of fuel carried at the beginning of the lap (float; strictly less than 1)
- the time [seconds] for a pit stop taking on no fuel (float)
- the extra pit stop time per litre of fuel loaded (float)

The Output:

For each input line, print the following information

- one line containing the seven input numbers in order
- one line containing three integers:
  - total race time
  - the amount of fuel to be loaded initially
  - the number of pit stops
- for each pit stop, a line containing:
  - the number of laps completed since the start of the race at the time of the pit stop
  - the amount of fuel to be taken on

All floating point results should be printed to 6 significant figures. All numbers on a single line should be separated by a single space.

Sample Input:

```
3     100 0           10 0       20 0
3     100 0           10 .1      20 0
3     100 2           10 0       20 1
3     100 4           10 0       20 1
3     100 2           10 .1      20 1
```

Sample Output:

```
3 100 0 10 0 20 0
300 30 0
3 100 0 10 0.1 20 0
300 37.1742 0
3 100 2 10 0 20 1
410 20 1
2 10
3 100 4 10 0 20 1
480 10 2
1 10
2 10
3 100 2 10 0.1 20 1
422.469 23.4568 1
2 11.1111
```