2011 Consortium for Computing Sciences in Colleges
Programming Contest
Saturday, November 12th
Furman University
Greenville, SC

There are eight (8) problems in this packet. Each team member should have a copy of the problems. These problems are NOT necessarily sorted by difficulty. You may solve them in any order.

Remember input/output for the contest will be from `stdin` to `stdout`. `stderr` will be ignored. Do not refer to or use external files in your source code. Extra white space at the end of lines is ignored, but extra white space at the beginning or within text on a line is not ignored. An extra blank line of output is ignored, but blank lines at the beginning or between lines of text are not ignored.

Have Fun & Good Luck! ☺

Problem 1. SMILE! :-)

Problem 2. Look and Say

Problem 3. Magic Clocks

Problem 4. CCSC Lottery

Problem 5. A Mathematical Exercise

Problem 6. Curve Killer

Problem 7. Josephus

Problem 8. Word Count

Problem 1

SMILE! :-)

September 19, 1982 — On this day, a Carnegie Mellon University computer science professor named Scott E. Fahlman posted a smiley face **:-)** on an electronic message board.

Not just content to remain a frozen smirk, it has learned to wink **;-)** and stick out its tongue **:-p**. Sometimes it gets sad **:-(** or confused **:-S**, and sometimes it's scared **=:-0** but most of the time it's cheerful **:-D**.

As it grew older, it had its first kiss **:-*** and smoked its first cigarette **:-Q** and even grew its hair out **&:-)**. It's shown its fashion sense, too, sporting sunglasses **B-)** and baseball caps **d:-)**, and even the occasional bow tie **:-)8**.

Your job is to count the number of original smiley faces in a single line of input. That is, the number of times the substring "**:-)**" occurs.

Input

Input consists of a single string of length *n*, ($2 \leq n \leq 80$). Assume the last character in the file will always be an end of line character.

Output

Output the total times the substring "`:-)`" occurs on a given line of input. The output should be formatted exactly as shown below.

Sample Input

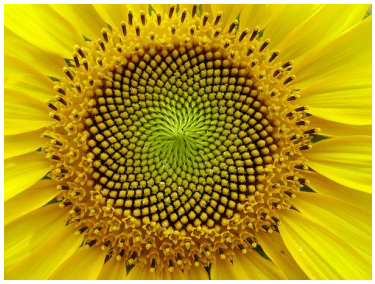
```
From: <Fahlman@Cmu>:-D :-) Read it sideways::-)-) ;-)Happy:-)8 Coding!
```

Output Corresponding to Sample Input

```
Smiley Count = 3
```

Problem 2

Look and Say



The look and say sequence is defined as follows. Start with any string of decimal digits as the first element in the sequence. Each subsequent element is defined from the previous one by "verbally" describing the previous element. For example, the string 122344111 can be described as "one 1, two 2's, one 3, two 4's, three 1's". Therefore, the element that comes after 122344111 in the sequence is 1122132431. Similarly, the string 101 comes after 1111111111.

Notice that it is generally not possible to uniquely identify the previous element of a particular element. For example, a string of 112213243 1's also yields 1122132431 as the next element.

Input & Output

The first line of input will be a number n, ($2 \leq n \leq 20$), telling how many cases are to follow. The next n-lines consist of a number of cases. Each case consists of a line of up to 1000 digits. For each test case, print the string that follows the given string. Assume all input will consist of decimal digits 0 – 9.

Sample Input

```
3
122344111
1111111111
12345
```

Output Corresponding to Sample Input

```
1122132431
101
1112131415
```

Problem 3

Magic Clocks

Andy is simply fascinated by clocks! In fact, he finds it hard to focus on his grading because he stares at his Mickey Mouse clock all day! So, to inspire him to grade his computer science programs he has devised a method to keep him on task. He will only look at the clock at "magical" times. A magical time is a time at which the angle between the hour hand and the minute hand is equal to the angle between the minute hand and the second hand on Andy's Disney clock.

Andy is so entranced by his clock that he just can't wait for the next magical time. Your task is to calculate the soonest magical time, given some time in the day. An important feature of Andy's analog clock is that it is *not continuous*, i.e. it does not smoothly flow into the next time. When the second hand ticks to 0, the minute hand *ticks* to the next minute, rather than continuously move. This is also the case with hours.

Input

The first line of input will be a number n, ($2 \leq n \leq 20$), telling how many lines of input are to follow. The next n-lines will contain a time in the `HH:MM:SS` format at the start of the line.

Output

For each time input, you should output "`The next magical time is s seconds away.`" If the time is already magical, then you should print, "`Look Andy, it's a magical time!`" If s is exactly one, print `second` rather than `seconds`. You may assume the next magical time is guaranteed to be within the next minute.

Sample Input

```
5
12:00:00
12:00:01
06:14:39
07:29:34
01:59:00
```

Output Corresponding to Sample Input

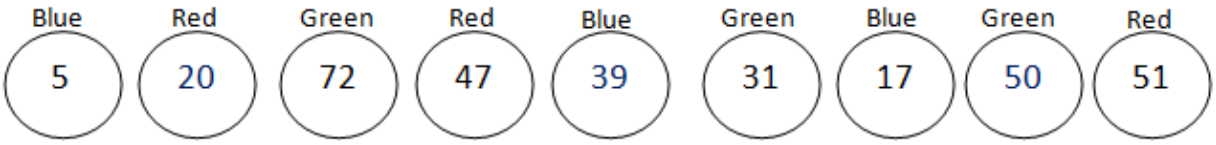
```
Look Andy, it's a magical time!
The next magical time is 59 seconds away.
The next magical time is 19 seconds away.
The next magical time is 1 second away.
The next magical time is 5 seconds away.
```

Problem 4

CCSC Lottery

This year CCSC has decided to find the winning numbers of the CCSC Lottery using a computer program. The winning numbers will be drawn from a bowl of balls. Each ball in this bowl has a color and a number. To decide the winning numbers, first we will construct chains of balls. Each chain should contain a ball from each color. That means if we have only three colors, then each chain should contain exactly three balls (one from each color).

For each such chain, the *value* of the chain is defined as the maximum difference between the numbers on the balls in the chain. For example, if we have a chain with three balls: Red with number 10, Green with 15, and Blue with 17, then the value of the chain will be 7 which is the maximum difference between 10, 15, and 17. Now the winning numbers should be constructed using the chain with the minimum value. The winning numbers should be written as the numbers from the balls in the chain in ascending order. Consider the following example.



Some of the chains and their corresponding values for this example will be as shown below.

5 (Blue) – 20 (Red) – 31 (Green) : Value 26
17 (Blue) – 20 (Red) – 31 (Green) : Value 14
20 (Red) – 31 (Green) – 39 (Blue) : Value 19
31 (Green) – 39 (Blue) – 47 (Red) : Value 16
39 (Blue) – 47 (Red) – 50 (Green) : Value 11

The last chain has the minimum value. So the winning number will be 39 47 50. You can assume that there will be exactly one chain with the minimum value.

Input
The first line of input will be n and m where n is the number of colors and m is the number of balls in each color. The next $n * m$ lines will describe each ball in the input in the following format.

Number *Color*

Number and *Color* will contain 1 more spaces between them. To simplify the input, each color will be represented by a number starting with 0.

Output
Your program should output the winning numbers of the one winner. Write the numbers from the chain with minimum value in ascending order. Separate each number by a single blank space.

Sample Input

```
3     3
5     0
20    1
72    2
47    1
39    0
31    2
17    0
50    2
51    1
```

Output Corresponding to Sample Input

```
39 47 50
```

Problem 5

A Mathematical Exercise

Professor X has made a New Year's resolution to get into better shape. To that end, he decides to spend his lunch hour running on a treadmill. Prof. X soon discovers that running in place is quite boring and he searches for a math problem to reduce the tedium. One day while running, he notices that sometimes the time elapsed and miles traveled are similar. For example, after running a minute and sixteen seconds, the digital readout for time elapsed is 1:16, while the distance traveled reading is 0.116 miles. He wonders how often this type of similarity occurs during his runs. He defines the readings as similar if:

```
(minutes elapsed) + (seconds elapsed)/100 = 10 × (miles traveled)
```

where `miles traveled = speed × (time elapsed)`. Your task is to find all similar readings.

Input

The first line of input is an integer T, ($2 \leq T \leq 20$), representing the number of test cases. Each test case will begin with n, ($1 \leq n \leq 20$), the number of segments during the run. The next n-lines of input will each correspond with one segment. A segment will consist of a line with two inputs, s and d, separated by a single space. The first s is a floating-point value representing Prof. X's running speed s in miles per hour for duration of d seconds where d is a positive integer. Prof. X never runs at the same speed for more than 30 minutes.

Output

For each test case, output the run number followed by, in ascending order, the times rounded to four decimal places when a similar reading occurred. If the readings are similar throughout an interval, only report the beginning time followed by an asterisk. Note that there is a special speed which is 3.6 mph. If the readings are similar and the speed of the treadmill is 3.6 mph, then the readings will remain similar until the start of the next minute. Always assume the clock resets to zero when we start a new run.

Sample Input

```
3
2
5.0 90
8.0 90
2
6.0 60
3.6 300
1
8.0 900
```

Output Corresponding to Sample Input

```
Run #1: 0:00.0000, 1:34.0909, 2:06.8182
Run #2: 0:00.0000, 1:00.0000*
Run #3: 0:00.0000
```

Problem 6

Curve Killer

Bob has finally graded his assembly tests and is passing them back out to his class. Unfortunately (but unsurprisingly), the grade average is much lower than it should be. Being the generous professor he is, he informs his students that he will curve the grades. This is how he will do so. Given the set of grades, we can calculate the average grade as follows:

$$\mu = \frac{\sum_{i=0}^{n} x_i}{n}$$

We can also calculate the standard deviation of the set in this manner.

$$\sigma = \sqrt{\frac{\sum_{i=0}^{n} (x_i - \mu)^2}{n-1}}$$

Given a new standard deviation and mean, we can scale the points such that the data set's mean and standard deviation change to this new value. Your job is to take the data set, and alter it such that the standard deviation and mean are changed to the new provided values.

Input

The input will consist of multiple test cases of the problem beginning with a line containing a number *n*, ($0 \leq n \leq 20$), representing the number of students for that case. The next line will contain two floating point numbers separated by a single space, the new mean and standard deviation in that order. This is followed by *n* lines each containing a student's integer grade. The last line will contain an *n* value of 0. Do not process this case.

Output

For each test case, you should print out each new grade on a new line. Print the grades in sorted order for each case. The sorted lists for each case should appear next to each other. Furthermore, print them as integers without rounding, i.e., 98.9 truncates to 98. Lastly, grades with values below zero are not permitted. If a grade falls below zero, it must be printed as zero.

Sample Input

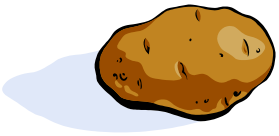
```
3
75.0 10.0
42
55
57
3
80.0 5.0
23
90
27
0
```

Output Corresponding to Sample Input

```
63
79
81
76
77
85
```

Problem 7

Josephus



The Josephus problem is the following game. There are *n* people, numbered 1 to *n*, sitting in a circle. Starting at person 1, a hot potato is passed. After *m* passes, the person holding the hot potato is eliminated, the circle closes ranks, and the game continues with the person who was sitting after the eliminated person picking up the hot potato. The last remaining person wins.

The Josephus problem arose in first century A.D., in a cave on a mountain in Israel, where Jewish zealots were besieged by Roman soldiers. The zealots voted to form a suicide pact rather than surrender to the Romans. Josephus suggested the game, and rigged the game so he would get the last lot. That is how we know about this game; in effect Josephus cheated!

Write a program that takes as input the number of people in the circle, *n*, and the number of passes, *m*. Display the sequence of people removed. The final person printed is deemed the winner. Note that person 1 is always the designated first person, and starts with the hot potato. You may assume the number of passes remains constant for each test case, and once person number *k* is eliminated, then the next round begins with person *(k+1)* holding the hot potato.

Input
The first line of input is an integer *T*, ($2 \leq T \leq 20$), representing the number of test cases. This is followed by *T* lines, each containing a test case with two integers separated by a single space. The first integer represents *n*, the number of people where ($2 \leq n \leq 40$), and the second represents *m*, the number of passes where ($0 \leq m \leq 50$).

Output
Your program should print the sequence of *n* people removed. Each person should be separated by a single space.

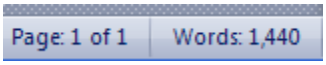
Sample Input
```
5
5 0
5 2
7 3
10 10
6 8
```

Output Corresponding to Sample Input
```
1 2 3 4 5
3 1 5 2 4
4 1 6 5 7 3 2
1 3 6 10 8 9 5 2 4 7
3 1 2 6 4 5
```

Problem 8

Word Count

Microsoft Word now automatically displays the number of words in your documents on the status bar at the bottom of the workspace.



You can even click on Words from the status bar to find the total characters and lines in your file too. This is nothing new on the *Linux* operating system which has always included a word count command called `wc` that displays the number of *lines*, *words*, and *bytes* in a text file to the screen. These are each defined as follows.

- *lines* — total end of line characters in the file.
- *words* — total *tokens* in the file. A *token* starts and ends with a printable, non-whitespace character. A *whitespace character* is a space, tab, or end of line character.
- *bytes* — total characters in the file including all *whitespace characters*.

You have been asked to implement the classic `wc` *Linux* system command to calculate the total lines, words, and bytes as defined above.

Input

Your program should accept a single test case consisting of one or more strings of length *n*, ($2 \leq n \leq 80$). Input is terminated by the end of file, and the last character in the file will always be an end of line character.

Output

Output the total number of lines, words, and bytes exactly as shown below.

Sample Input

```
"Your time is limited, so don't waste it living someone else's life.
Have the courage to follow your heart and intuition.
They somehow already know what you truly want to become.
Everything else is secondary."
-Steve Jobs, 2005 Stanford Commencement
```

Output Corresponding to Sample Input

```
Lines = 5
Words = 40
Bytes = 250
```