# Problem A: Sixth Grade Math (white balloon)

In sixth grade, students are presented with different ways to calculate the Least Common Multiple (**LCM**) and the Greatest Common Factor (**GCF**) of two integers. The **LCM** of two integers ***a*** and ***b*** is the smallest positive integer that is a multiple of both ***a*** and ***b***. The **GCF** of two non-zero integers ***a*** and ***b*** is the largest positive integer that divides both ***a*** and ***b*** without remainder.

For this problem you will write a program that determines both the **LCM** and **GCF** for positive integers.

## Standard Input

The first line of input contains a single integer ***N***, ($1 \le N \le 1000$) which is the number of data sets that follow. Each data set consists of a single line of input containing two positive integers, ***a*** and ***b***,

($1 \le a,b \le 1000$) separated by a space.

## Standard Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space, the LCM, a space, and the GCF.

| Sample Input | Sample Output |
|---|---|
| 3 | 1 10 5 |
| 5 10 | 2 161 1 |
| 7 23 | 3 168 14 |
| 42 56 | |

# Problem B: Cryptoquote (blue balloon)

A cryptoquote is a simple encoded message where one letter is simply replaced by another throughout the message. For example:

|  |  |
|---|---|
| Encoded: | **`HPC PJVYMIY`** |
| Decoded: | **`ACM CONTEST`** |

In the example above, **H=A**, **P=C**, **C=M**, **J=O**, **V=N**, **Y=T**, **M=E** and **I=S**. For this problem, you will decode messages.

## Standard Input

The first line of input contains a single integer ***N***, ($1 \le N \le 1000$) which is the number of data sets that follow.
Each data set consists of two lines of input. The first line is the encoded message. The second line is a 26 character string of upper case letters giving the character mapping for each letter of the alphabet: the first character gives the mapping for **A**, the second for **B** and so on. Only upper case letters will be used. Spaces may appear in the encoded message, and should be preserved in the output string.

## Standard Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space and the decoded message.

| Sample Input | Sample Output |
|---|---|
| 2 | 1 ACM CONTEST |
| HPC PJVYMIY | 2 THE SKY IS BLUE |
| BLMRGJIASOPZEFDCKWYHUNXQTV |  |
| FDY GAI BG UKMY |  |
| KIMHOTSQYRLCUZPAGWJNBVDXEF |  |

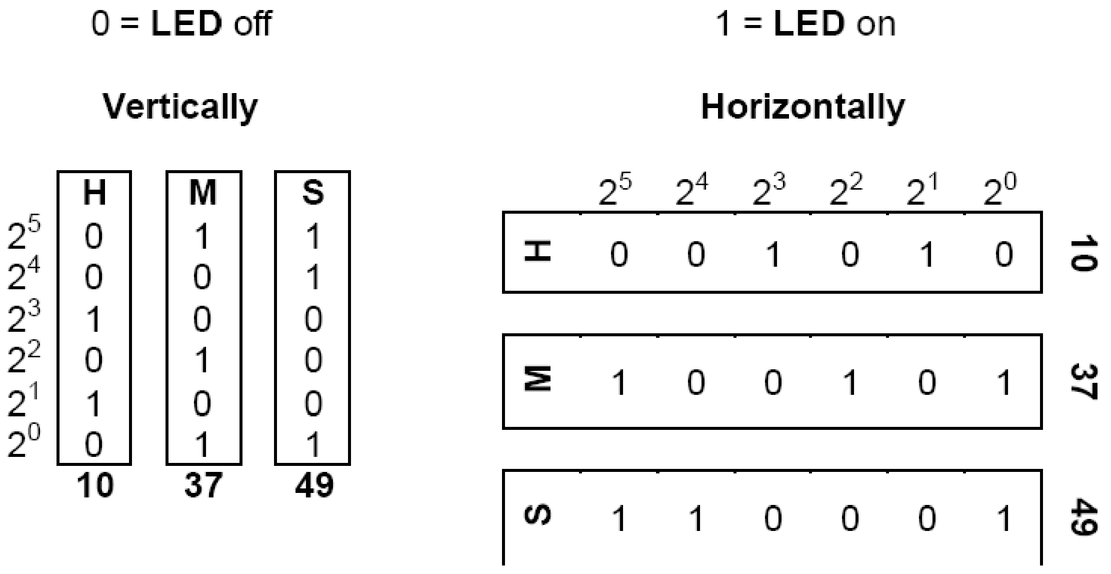
## Problem C: Binary Clock (green balloon)

A binary clock is a clock which displays traditional sexagesimal time (military format) in a binary format. The most common binary clock uses three columns or three rows of LEDs to represent zeros
and ones. Each column (or row) represents a time-unit value.

When three columns are used (vertically), the bottom row in each column represents **1** (or **$2^0$**), with each row above representing higher powers of two, up to **$2^5$** (or **32**). To read each individual unit (hours, minutes or seconds) in the time, the user adds the values that each illuminated LED represents, and then reads the time from left to right. The first column represents the hour, the next column represents the minute, and the last column represents the second.

When three rows are used (horizontally), the right column in each row represents **1** (or **$2^0$**), with each column left representing higher powers of two, up to **$2^5$** (or **32**). To read each individual unit (hours, minutes or seconds) in the time, the user adds the values that each illuminated LED represents, and then reads the time from top to bottom. The top row represents the hour, the next row represents the minute, and the bottom row represents the second.

For example:



Time is: **10 : 37 : 49**

For this problem you will read a time in sexagesimal time format, and output both the vertical and horizontal binary clock values. The output will be formed by concatenating together the bits in each column (or row) to form two 18 character strings of 1's and **0**'s as shown below.

**10:37:49** would be written vertically as **011001100010100011** and horizontally as **001010100101110001**.

## Standard Input

The first line of input contains a single integer ***N***, (1 ≤ ***N*** ≤ 1000) which is the number of data sets that follow. Each data set consists of a single line of input containing the time in sexagesimal format.

## Standard Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space, the binary time in vertical format (18 binary digits), a space and the binary time in horizontal format (18 binary digits).

| Sample Input | Sample Output |
|---|---|
| `2` | `1 011001100010100011 001010100101110001` |
| `10:37:49` | `2 000000000000000001 000000000000000001` |
| `00:00:01` | |

## Problem D: Recursively Palindromic Partitions

## (yellow balloon)

A *partition* of a positive integer ***N*** is a sequence of integers which sum to ***N***, usually written with plus signs between the numbers of the partition. For example

```
15 = 1+2+3+4+5 = 1+2+1+7+1+2+1
```

A partition is *palindromic* if it reads the same forward and backward. The first partition in the example is *not* palindromic while the second is. If a partition containing ***m*** integers is palindromic, its left half is the first **`floor(m/2)`** integers and its right half is the last **`floor(m/2)`** integers (which must be the reverse of the left half). (**`floor(x)`** is the greatest integer less than or equal to ***x***.)

A partition is *recursively palindromic* if it is palindromic and its left half is recursively palindromic or empty. Note that every integer has at least two recursively palindromic partitions one consisting of all ones and a second consisting of the integer itself. The second example above is also recursively palindromic.

For example, the recursively palindromic partitions of 7 are:

```
7, 1+5+1, 2+3+2, 1+1+3+1+1, 3+1+3, 1+1+1+1+1+1+1
```

Write a program which takes as input an integer ***N*** and outputs the number of recursively palindromic partitions of ***N***.

### Standard Input

The first line of input contains a single integer ***N***, ($1 \le N \le 1000$) which is the number of data sets that follow. Each data set consists of a single line of input containing a single positive integer for which the number of recursively palindromic partitions is to be found.

### Standard Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), a space and the number of recursively palindromic partitions of the input value.

| Sample Input | Sample Output |
|---|---|
| 3 | 1 4 |
| 4 | 2 6 |
| 7 | 3 60 |
| 20 | |

## Problem E: Text Messaging Improvement?

## (brown balloon)

On a standard mobile phone the letters are distributed across the keys 2 through 9 as:

| | | |
|---|---|---|
| | **2** **ABC** | **3** **DEF** |
| **4** **GHI** | **5** **JKL** | **6** **MNO** |
| **7** **PQRS** | **8** **TUV** | **9** **WXYZ** |

To enter the letter **C**, you press key 2 three times (seeing **A**-**B**-**C**). The number of keystrokes to enter a letter depends on where it is in the list of letters on its key.

The *Flathead Telephone Company* (FTC) is considering rearranging the letters on the keys to reduce the average number of keystrokes required to enter names etc. or send text messages. The letters must still appear in alphabetical order on the keys but different numbers of letters may appear on each key and possibly more keys could be used. FTC has several databases of letter frequencies used in different applications. For instance, it might help to move **S** from the **7** key to the **8** key. They need a program which is given the frequencies of the letters and a number of keys and returns the assignment of letters to keys with the smallest average number of keystrokes using the given frequencies. Each key used must have at least one letter and at most eight letters.

### Standard Input

The first line of input contains a single integer ***N***, ($1 \leq N \leq 1000$) which is the number of data sets that follow. Each data set consists of three lines of input. The first line contains a single integer ***K***, ($4 \leq K \leq 26$), the number of keys which are to be used. The second and third lines contain 13 decimal values each giving the percent frequency of the letters **A** through **Z** in order.

## Standard Output

For each data set, you should generate one line of output with the following values: The data set number as a decimal integer (start counting at one), the best average number of keystrokes to three decimal places, a space and the letters **A** through **Z**, for the best arrangement, in order with a single

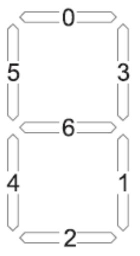
**Sample Input**

```
2
8
8.167 1.492 2.782 4.253 12.702 2.228 2.015 6.094 6.966 0.153 0.772 4.025 2.406
6.749 7.507 1.929 0.095 5.987 6.327 9.056 2.758 0.978 2.360 0.150 1.974 0.075
9
1.0 10.0 11.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
10.0 10.0 10.0 10.0 10.0 11.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
```

**Sample Output**

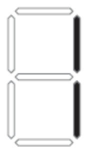
```
1 1.647 AB CD EFG HIJK LM NOPQ RS TUVWXYZ
2 1.570 A B CDEFG HIJKLM N OP QR STUV WXYZ
```

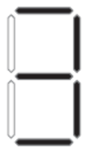
# Problem F: Adding Sevens (gray balloon)

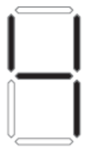


**Description**

A seven segment display, similar to the one shown on the right, is composed of seven light-emitting elements. Individually on or off, they can be combined to produce 127 different combinations, including the ten Arabic numerals. The figure on the next page illustrates how the ten numerals are displayed. 7-seg displays (as they're often abbreviated) are widely used in digital clocks, electronic meters, and calculators. A 7-seg has seven connectors, one for each element, (plus few more connectors for other electrical purposes.) Each element can be turned on by sending an electric current through its pin. Each of the seven pins is viewed by programmers as a single bit in a 7-bit number, as they are more comfortable dealing with bits rather than electrical signals. The figure on the right shows the bit assignment for a typical 7-seg, bit 0 being the right-most bit. For example, in order to display the digit 1, the programmer knows that only bits 1 and 3 need to be on, i.e. the 7-bit binary number to display digit 1 is "0001010", or 10 in decimal. Let's call the decimal number for displaying a digit, its *display code,* or just *code* for short. Since a 7-seg displays 127 different configurations, display codes are normally written using 3 decimal places with leading zeros if necessary, i.e. the display code for digit 1 is written as 010.


0001010
code: 010


1001111
code: 079


1101010
code: 106

In a 9-digit calculator, 9 7-seg displays are stacked next to each other, and are all controlled by a single controller. The controller is sent a sequence of $3n$ digits, representing $n$ display codes, where $0 < n < 10$. If $n < 9$, the number is right justified and leading zeros are automatically displayed. For example, the display code for 13 is 010079 while for 144 it is 010106106

Write a program that reads the display codes of two numbers, and prints the display code of their sum.

**Input Format**

Your program will be tested on one or more test cases. Each test case is specified on a single line in the form of `A+B=` where both `A` and `B` are display codes for decimal numbers $a$ and $b$ respectively where $0 < a, b < a + b < 1,000,000,000$. The last line of the input file is the word `"BYE"` (without the double quotes.)

**Output Format**

For each test case, print `A+B=C` where `C` is the display code for $a + b$.

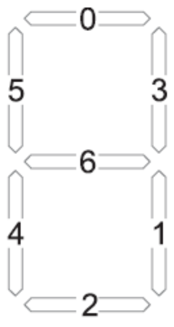
**Sample Input/Output**

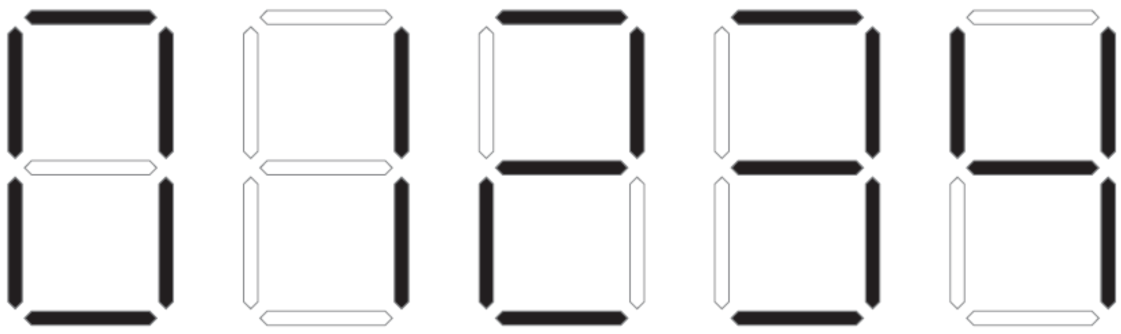
INPUT

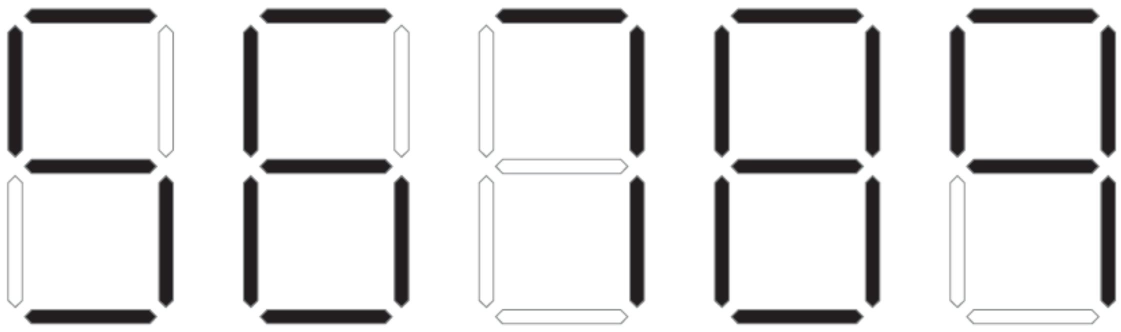
```
010079010+010079=
106010+010=
BYE
```

OUTPUT

```
010079010+010079=010106106
106010+010=106093
```


0
5
3
6
4
1
2

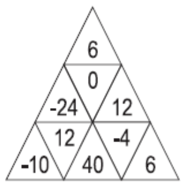




# Problem G: Adding up Triangles

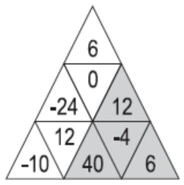
# (purple balloon)

## Description



Take a look at the triangle on the right. It is made of 9 (unit) triangles arranged in three rows (N=3). Needless to say, a unit triangle is a triangle with N=1. If you study the figure for few seconds, you'll realize that you can find 13 different triangles (which we'll call sub-triangles.) Of these 13 sub-triangles we have: Nine unit triangle; three with N=2, and one with N=3. The following table lists the number of sub-triangles in arrangements with $N < 5$.

| | | | | |
|---|---|---|---|---|
| **# of Rows:** | $N = 1$ | $N = 2$ | $N = 3$ | $N = 4$ |
| **# of Sub-triangles:** | 1 | 5 | 13 | 27 |



Let's define the value of a unit triangle to be the integer value written in that triangle. In general, the value of a triangle is the sum of values in all its unit triangles. The figure on the right is the same as the one above but with the sub-triangle having the largest value being highlighted. Write a program to determine the sub-triangle with the largest value.

## Input Format

Your program will be tested on one or more test cases. Each test case is specified in a single line made of integers (separated by spaces.) The first integer is the number of rows in the test case, and the remaining integers are the values of the unit triangles specified in a top-down, left-to-right order. (the first test case in the example below is the same as the one in the figure.) The last line of the input file contains the number 0 (which is not part of the test cases.)

The maximum number of rows is 400. The absolute value of a unit triangle is less than 1000.

## Output Format

For each test case, print the result using the following format:

```
k.␣V
```

where k is the test case number (starting at 1,) ␣ is a single space, and V is the maximum value of a sub-triangle in that test case.

## Sample Input/Output

INPUT

```
3 6 -24 0 12 -10 12 40 -4 6
4 1 1 -1 1 1 -1 1 -1 1 1 -1 1 -1 1 -1 1
0
```

OUTPUT

```
1. 54
2. 4
```

# Problem H: Relax! It's just a game (red balloon )

**You:** What's the score? Did I miss much?

**Me:** It's 2-1 for **DRAGONS** and the second half just started. The first half was quite boring.

**You:** Who scored first? **DRAGONS** or **MOGAS 90?**

**Me:** What difference does it make?

**You:** Big difference! I can predict the outcome of the match if I knew the order of which goals were scored in the first half.

**Me:** What do you mean?

**You:** It's 2-1 for **DRAGONS**, right? One of three things could have happened: **DRAGONS** scored two goals then **MOGAS 90** scored; Or, **DRAGONS** scored its first goal, then **MOGAS 90**, then **DRAGONS** again; Or, **MOGAS 90** scored first, then **DRAGONS** scored its two goals.

**Me:** *So?!!* I still don't understand what difference does that make? It's still 2-1 for **DRAGONS**! Why don't you just relax and let us continue watching the game in peace.

**You:** *You don't understand!!* I believe the probability of who'll win depends on the order of how goals were scored. Now I have to predict the outcome for 3 possibilities.

**Me:** And what if the score was 3-2? What would you have done then?

**You:** I would have to work for 5 different possibilities. No?

**Me:** *Of course not!* The number of possibilities isn't always equal to the sum.

**You:** Can you tell me when will it be equal to the sum?

**Me:** You're a programmer, why don't you write a program that counts the number of possibilities and compare it to the sum?

**You:** I don't have the time, I want to watch the match. Besides, I have nine other problems to worry about.

**Me:** I'll give you a hint. The possibilities will be equal to the sum only if one of the teams scored a certain number of goals

## Standard Intput

Your program will be tested on one or more test cases. Each test case specifies two natural numbers (**A** and **B**) (separated by one or more spaces) representing the score of the first half.

No team will be able to score more than 10 goals. The last line of the input file contains two -1's (which is not part of the test cases.)

## Standard Output

For each test case where the number of possibilities is equal to the sum, print: **A+B=C** Where **A** and **B** are as above and **C** is their sum. If the number of possibilities is not equal to the sum, replace the **'='** sign with **'!='** (without the quotes.)

**Sample Input/Output**

INPUT

```
2 1
1 0
-1 -1
```

OUTPUT

```
2+1=3
1+0=1
```

# Problem I: Think I'll buy me a football team! (pink balloon )

## Description

Falling Stocks. Bankrupted companies. Banks with no Cash. Seems like the best time to invest: *"Think I'll buy me a football team!"*
No seriously, I think I have the solution to at least the problem of cash in banks. Banks nowadays are all owing each other great amounts of money and no bank has enough cash to pay other banks' debts even though, on paper at least, they should have enough money to do so. Take for example the inter-bank loans shown in figure (a). The graph shows the amounts owed between four banks (A. . . D). For example, A owes B 50M while, at the same time, B owes A 150M. (It is quite common for two banks to owe each other at the same time.) A total amount of 380M in cash is needed to settle all debts between the banks.
In an attempt to decrease the need for cash, and after studying the example carefully, I concluded that there's a lot of cash being transferred unnecessarily. Take a look:

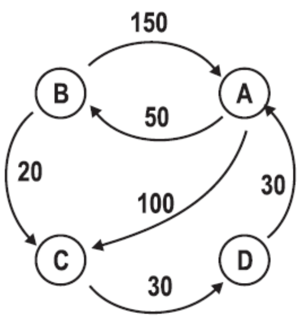


Figure (a)

1. C owes D the same amount as D owes A, so we can say that C owes A an amount of 30M and get D out of the picture.
2. But since A already owes C 100M, we can say that A owes C an amount of 70M.
3. Similarly, B owes A 100M only, (since A already owes B 50M.) This reduces the above graph to the one shown in figure (b) which reduces the needed cash amount to 190M (A reduction of 200M, or 53%.)
4. I can still do better. Rather than B paying A 100M and A paying 70M to C, B can pay 70M (out of A's 100M) directly to C. This reduces the graph to the one shown in figure (c). Banks can settle all their debts with only 120M in cash. A total reduction of 260M or 68%. *Amazing!*

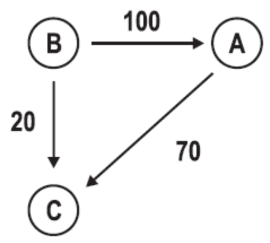


Figure (b)

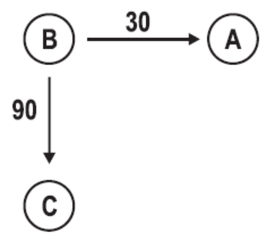


Figure (c)

I have data about inter-bank debts but I can't seem to be able to process it to obtain the minimum amount of cash needed to settle all the debts. Could you please write a program to do that?

### Input Format

Your program will be tested on one or more test cases. Each test case is specified on $N + 1$ lines where $N < 1,000$ is the number of banks and is specified on the first line. The remaining $N$ lines specifies the inter-bank debts using an $N \times N$ adjacency matrix (with zero diagonal) specified in row-major order. The ith row specifies the amounts owed by the ith bank. Amounts are separated by one or more spaces. All amounts are less than 1000.

The last line of the input file has a single 0.

### Output Format

For each test case, print the result using the following format:

```
k.␣B␣A
```

where k is the test case number (starting at 1,) ␣ is a space character, B is the amount of cash needed before reduction and A is the amount of cash after reduction.

### Sample Input/Output

INPUT

```
4
  0  50 100   0
150   0  20   0
  0   0   0  30
 30   0   0   0
0
```

OUTPUT

```
1. 380 120
```

# Problem J: A day at the Races (black balloon )

## Description

*Formula One* is the highest class of car racing sports. A typical *Formula One season* consists of a series of races called *"Grands Prix"* which constructors like Ferrari, Renault, etc. and others participate with one or more cars driven by the best drivers in the world. During the season, teams compete in two parallel championships: the *drivers championship* and the *teams championship.*

In the **drivers championship,** drivers compete to achieve the maximum total number of points by the end of the season, the rules of the competition states that the top eight drivers at each Grand Prix receive 10,8,6,5,4,3,2,1 points respectively. In case of points tie, the driver with the highest number of first places leads. If still tied, then the highest second places, and so on till the highest 8th places. If still tied, then drivers are sorted lexicographically by their last and then by their first names.

After each race, the points received by each driver are added to his team's pocket, and at the end of the season the team with the highest number of points wins the **teams championship.** To add excitement to the season, team sponsors are allowed to buy drivers from other teams even within the same season. In case of points tie between teams, teams are sorted lexicographically by their names. In this problem, you are given data of a formula one season and you're asked to process these data according to the rules above to determine both the drivers and teams standings.

## Input Format

Your program will be tested on one or more data-sets, each representing a Formula One season. All input lines are 255 characters or less. Studying the sample I/O you'll discover that the first line of each season has an integer $N$, where $0 < N < 32$ and representing the number of Grands Prix in that season. For each Grand Prix, the name of the Grand Prix appears on a line by itself (maximum length is 64 characters) followed by a table of the first name, last name and team name of the top eight drivers, from 1 to 8, in that Grand Prix. Each of the first and last names is a sequence of printable ASCII characters, no longer than 12 characters, and contains no spaces. Each team name is a sequence of printable ASCII characters, no longer than 18 characters, and may contain spaces (but no leading or trailing spaces.) Each team name is followed by a single period `'.'` which is not part of the name. Trailing white space may follow. A line of three `-`'s follows the listing of each Grand Prix. The last line of the input file contains a single zero.

## Output Format

For each data set in the input you must print "`Season k:`" where `k` is the data-set number (starting from 1.) The next line must state "`Drivers Standing:`". On subsequent lines list the drivers standing for that season. For each driver, print their first and last names separated by exactly one space and left justified in a field of width 25, followed by a single space, followed by the total number of points achieved by the driver during the season. The drivers standing should be followed by a blank line.

The next line must state "`Teams Standing:`" On subsequent lines list the teams standing for the that season. For each team, print the team name left justified in a field of width 25, followed by a single space, followed by the total number of points the team has scored during the season. The teams standing should be followed by a blank line.

**Sample Input/Output**

INPUT

```
2
FORMULA 1 Gran Premio Telefonica de Espana 2006
Pos  Driver                    Team
1    Fernando Alonso           Renault.
2    Michael Schumacher        Ferrari.
3    Giancarlo Fisichella      Renault.
4    Felipe Massa              Ferrari.
5    Kimi Raikkonen            McLaren-Mercedes.
6    Jenson Button             Honda.
7    Rubens Barrichello        Honda.
8    Nick Heidfeld             Sauber-BMW.
---
FORMULA 1 Grand Prix de Monaco 2006
Pos  Driver                    Team
1    Fernando Alonso           Renault.
2    Jaun-Pablo Montoya        McLaren-Mercedes.
3    David Coulthard           RBR-Ferrari.
4    Rubens Barrichello        Honda.
5    Michael Schumacher        Ferrari.
6    Giancarlo Fisichella      Renault.
7    Nick Heidfeld             Sauber-BMW.
8    Ralf Schumacher           Toyota.
---
0
```

OUTPUT

```
Season 1:
Drivers Standing:
Fernando Alonso         20
Michael Schumacher      12
Giancarlo Fisichella    9
Jaun-Pablo Montoya      8
Rubens Barrichello      7
David Coulthard         6
Felipe Massa            5
Kimi Raikkonen          4
Jenson Button           3
Nick Heidfeld           3
Ralf Schumacher         1

Teams Standing:
Renault                 29
Ferrari                 17
McLaren-Mercedes        12
Honda                   10
RBR-Ferrari             6
Sauber-BMW              3
Toyota                  1
```