The “Assorted Stuff” Programming Contest

Cornell ACSU Programming Contest, Fall 2003

October 18, 2003

Welcome to the annual ACSU programming contest. We have prepared six problems of various difficulty for you to solve. This year, there is no common theme behind the problems. Instead, you get to enjoy the literary qualities of different judges (and you are free to guess who wrote which problem).

The allowed programming languages are C/C++ and Java. You should submit your source code file using the PC^2 software, making sure to follow the naming conventions for your source code, and the input file you read.

Note that your program should read its input from the respective input file, while its output should go to `stdout`.

As usual, you may not use any electronic devices besides the lab computer, and may not open any web browser for whatever reason.

Many of the problems are specified by a bunch of numbers. Whenever that is the case, the numbers will be separated by whitespace and/or end-of-line characters, but no tabs. The amount of whitespace as well as where exactly line breaks occur is not specified — this should not make your programming task any more complicated.

One warning that seems to be in place after past contests: all of our numbering (input data sets, persons, etc.) always starts at 1, not at 0.

And a piece of advice: several problems ask you to print numbers rounded to two decimals. Here's how to do it:

- In C, you do this via `printf ("%.2f", r);` where `r` is the variable you want to print.
- in C++, the right thing to say is `cout.precision(2); cout << fixed << r;`
- In Java, the syntax is `System.out.print ((new java.text.DecimalFormat("#.##")).format(r));`

[Including this page, this problem set should contain 5 double-sided pages. If yours doesn't, please contact one of the helpers immediately.]

Problem A: Bloody Quartet

File Name: `bloody.cpp|bloody.java`

Input File: `bloody.in`

Having robbed seventeen banks, fifteen post offices, thirty two jewelery stores, two trains and one almost deaf old lady, the Bloody Quartet was one of the most successfull group of robbers of the Wild West.

At the end of their fruitful career, not knowing what to do with all the money and stuff accumulated over the years, they decided to put it all into a large treasure chest and store it for future generations. Together they picked a secret location O, at which they decided to bury the chest. They dug a deep hole in the ground (which was not an easy task, considering that the best tools available to them were a set of French silver spoons from the 17-th century) and buried the treasure box in it. And then they parted. In order not to raise suspicion, they decided to leave in different directions, at right angles to each other. And so they did. Each of them went off in the chosen direction, and lived happily ever after (or until caught, imprisoned, stoned, hanged, shot and dragged by horses around Sheriff's office, that is).

Description

For you as a researcher, the chest would have an immense historical value (not to mention the value of the gold inside). Unfortunately, the only reliable information you have are reports of sightings of the four robbers. Each robber was seen once, and for each robber i you know the point P_i where i was seen ($i = 1, \ldots, 4$). Your conjecture is that the secret location O is such that it is possible to shoot four rays from O in four directions with right angles between them, each ray will pass through one of the points P_1 through P_4 (and each point will be hit by a ray). Your program is either to output the coordinates of the location O, or report that no such location exists (based on the four points given).

Input

The first line contains a single integer K − the number of data sets. Each data set consists of four lines, each containing two real numbers separated by a space. The numbers x_i, y_i on the i-th line are cartesian coordinates of the i-th robber. Each coordinate is given with precision to three decimal places and is at most 1000 in absolute value. There is an empty line following each data set.

Output

For each data set, output "`Data set` i`:`", where i is the number of the set, followed by a space and the answer to the data set. The answer should be either the x- and y-coordinates of the treasure rounded to two decimal places and separated by a space, or "`Impossible`" if the coordinates of robbers do not determine a position of the treasure consistent with the problem description.

Sample Input/Output

Sample input `bloody.in`

```
2
82.000 23.000
7.000 22.000
2.000 -9.000
31.000 -38.000

3.025 5.025
9.0025 14.025
-6.975 -9.975
3.025 5.025
```

Corresponding output

```
Data set 1: 17.00 -3.00
Data set 2: 3.025 5.025
```

Problem B: Formatting

File Name: `format.cpp|format.java`

Input File: `format.in`

Description

Your task is to write a simple formatting tool that processes text documents consisting of printable characters (alphanumberic characters, commas, dots, hyphens), and whitespace (spaces and newlines). Each paragraph should be formatted to fit a certain width, paragraphs should be separated by a blank line, and extra whitespace should be removed.

Each document consists of *words*. For our purposes, a word is just a sequence of printable characters delimited by whitespace. A *paragraph* is a sequence of words with whitespace (that is, spaces and/or newlines) in between. An empty line (containing no printable characters) denotes the end of a paragraph. The end of a document is denoted by a single character "`*`" on a separate line.

For each document, you will be given its desired *width* W. Your task is process the input document and output it in a formatted form. The lines of the input document may begin with spaces, and there may be multiple spaces between words or even at the end of the line. There may be multiple empty lines before or after each paragraph. The length of lines may be arbitrary, however, no word exceeds 80 characters in length.

The formatted document should meet the following specifications.

1. It should contain no empty lines at the beginning or at the end, and exactly one empty line between every two consecutive paragraphs.
2. Each paragraph should be formatted into lines of length at most W. Fit as many words on each line as possible, while not exceeding the maximum length. If a single word has length greater than W, it should be on a line by itself.
3. There should be exactly one space between consecutive words of each line. There should be no spaces at the beginning or end of each line.

Input

The first line of each input file contains a single integer K – the number of documents to process. K document descriptions follow. Each document description begins with the desired width W ($2 \leq W \leq 80$) on a separate line, followed by the body of the unformatted document. Each document is terminated by a single "`*`" on a separate line.

Output

For each document in the input file, output a line with the text "`Data set` i:", where i is the number of the document. Then output the formatted document itself (without the terminating `*`). Output an extra blank line after each document.

Sample Input/Output

(Turn the page for an example.)

Sample input `format.in`

```
3
12
This is how your formatted text
    should    look
 like.

Good luck.
*

80
*

10
Some very-long-words must
be on a separate line.


    A
  nother
   para
  graph.

*
```

Corresponding output

```
Data set 1:
This is how
your
formatted
text should
look like.

Good luck.

Data set 2:

Data set 3:
Some
very-long-words
must be on
a separate
line.

A nother
para
graph.
```

Problem C: Perpetual TA

File Name: `perpetual.cpp|perpetual.java`

Input File: `perpetual.in`

Description

Those silly undergrads, always turning in their assignments late. Tracy the 211 TA is left with the unfortunate job of grading all these assignments. It takes her some constant amount of time to grade each paper, but everyday it seems more and more are coming in. To make matters worse for poor Tracy, she has deadlines herself and must finish grading all the assignments by a set date. Before this date though she is grading constantly, food and sleep mere luxuries. Help her out to determine if her task is at all possible, or if she should just give up now.

Input

First number K is number of data sets to follow. Each data set begins with two integers N and H. N $(0 < N \leq 100)$ is the number of days from the start of when assignments are received to when all grades are due and H $(0 \leq H \leq 24)$ the number of hours it takes Tracy to grade one assignment. Following this are N integers n_1, n_2,..., n_N, where n_i is the number of assignments Tracy receives to grade on the i-th day. Assume that assignments are received exactly at the beginning of each day, and all grades are due by the end of the last day, each day being 24 hours.

Output

For each of the K data sets output "`Data set` x`: `" where x is the number of that data set, then either "`Yes, with` T `hours to spare`" if Tracy can grade all the assignments by the end of the last day and T is the maximum time she can have between finishing grading and the deadline, if she schedules things optimally. Otherwise, if she can not finish grading before the deadline output "`Impossible`".

Sample Input/Output

Sample input `perpetual.in`

```
2
5 5
5 5 5 5 4
5 5
5 5 5 5 5
```

Corresponding output

```
Data set 1: Yes, with 0 hours to spare
Data set 2: Impossible
```

Problem D: Raster Images

File Name: raster.cpp|raster.java

Input File: raster.in

Description

One way of storing images in files involves writing two separate files - a header file, and a raster file. The header file contains important information about the size of the file, and the type of color encoding used. The raster file contains all of the actual data. You have a number of images that are stored this way, but your little brother snuck onto your computer one night, and executed the command "`rm *.header`", deleting all of your header files. However, being a programmer, you figure that you can probably recover all the lost info from the missing header files. You remember that all of your images were encoded in 24 bit RGB color, meaning that, for each pixel in the image, there were three bytes representing the Red, Green, and Blue portions of that pixel. So, the only missing information is the dimensions of the images.

You know that the raster file was created by writing the data from each pixel, from left to right, and then top to bottom to a file. So, if the image had m rows, and n columns, then a pixel in row i, column j, would be represented by the three bytes starting at $3(n \times i + j)$. After some thought, you come up with a way to programmatically determine what the most likely dimensions were:

Since there are a limited number of factors of the size of the raster file, only a relatively small number of dimensions will work. Second, you guess that pixels that are adjacent in the image, will probably have similar colors. So you define a distance function between the colors of two pixels, which will provide a metric of how similar two colors are: $\sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$. Next, you consider every possible pair of dimensions that have at least 2 rows and 2 columns. For each pair of dimensions, you evaluate the average distance between all adjacent pixels down each column, with dimensions that give lower averages being favored.

The only problem with this method is that it tends to give you very wide, short images, or very tall narrow images. To avoid this, you decide to add some favoritism for images that are more nearly square. Thus, you multiply the average pixel distance by $\max(\text{rows}, \text{columns})$ so that extreme dimensions are disfavored. This gives you the overall score for a pair of dimensions. Your goal now is for each raster file, find the dimensions that give you the lowest score.

Input

The first line of the input contains a single number K – the number of data sets. K blocks follow, one for each test case. Each block starts with a line containing a non-prime number N, $4 \leq N \leq 10000$, – the number of pixels in the raster. N lines follow, each line i containing three integers R, G, B in the range $[0, 255]$, that specify the intensities of the red, green and blue color of the i-th pixel in the raster file. Each test case will be followed by a blank line.

Output

Output one line for each test case. It should begin with "`Data set` x:", where x is the number of the data set, and a space. Following this, output the optimal dimensions of the image and the score for these dimensions. More precisely, output three space separated numbers: the optimum width and height in pixels, and the optimum score, rounded to one decimal place. If multiple dimensions achieve optimum score, output the solution with minimum width.

Sample Input/Output

Sample input `raster.in`

```
2
12
118 51 40
42 103 68
65 40 48
124 32 58
66 85 36
41 40 83
71 32 32
66 86 35
40 90 95
103 102 106
107 51 98
118 35 34
4
65 65 65
65 65 65
65 65 65
65 65 65
```

Corresponding output

```
Data set 1: 3 4 78.22
Data set 2: 2 2 0.00
```

Problem E: Tiles

File Name: `tiles.cpp|tiles.java`

Input File: `tiles.in`

Description

One of the jury members of this contest caught a very rare disease: he started to tile dominoes. More precisely he picks some number N, draws a rectangle of size $3 \times N$ and tries to tile it up with dominoes (as you all know each domino has size 2×1). He says that his job will be finished after he counts all possible tilings including rotations and reflections. He already explored all tilings for $N = 4$ (see Figure 1) and now is eager to explore larger values of N. Hurrying to help their fellow, other jury members ask you to help them by writing the program which for given N will count all possible tilings of dominoes on rectangle $3 \times N$.

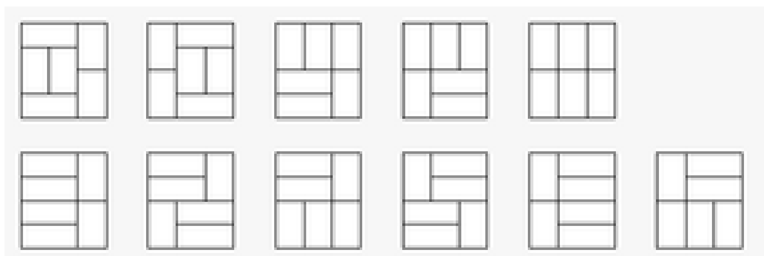

Figure 1: All possible tilings of a rectangle 3×4.

Input

The first line of the input file contains the number of test cases K. K lines follow, one for each test case. For each test case, the corresponding line contains a single number N ($1 \leq N \leq 30$) – the vertical dimension of the $3 \times N$ rectangle to be tiled.

Output

For each test case, output "`Data set` x`:`" the total number of tilings of the rectangle.

Note: You are guaranteed that total number of possible tilings will not exceed 1 000 000 000.

Sample Input/Output

Sample input `tiles.in`

```
5
4
5
8
10
12
```

Corresponding output

```
Data set 1: 11
Data set 2: 0
Data set 3: 153
Data set 4: 571
Data set 5: 2131
```

Problem F: Water

File Name: `water.cpp|water.java`

Input File: `water.in`

Description

The mad scientist Joseph Josephson is attempting to figure out what quantities of water he can measure out for an experiment. Joseph has four test tubes, each of which has capacity 10 milliliters and contains some quantity of water which is known to him.

Joseph can apply the following action to obtain different quantities of water: He can pour the water from one test tube (the source) into another test tube (the target). When he does this, he always fills the target as much as he can: he pours from the source into the target until either there is no water left in the source, or until the target is full.

Here's an example: let's say that Joseph starts off with his four test tubes having 3, 8, 4, and 0 milliliters of water. He can measure out 5 milliliters of water by pouring from the second tube to the first, so that his test tubes have water levels of 10, 1, 4, and 0 milliliters; and then, by pouring from the second tube to the third, so that his test tubes have water levels of 10, 0, 5, and 0 milliliters.

Input

The first number K is the number of data sets, followed by K lines of input. Each line contains exactly four integers between 0 and 10 inclusive, denoting the number of milliliters of water in each of his four test tubes.

Output

For each data set, first output "`Data Set` x`:`" on a line by itself, where x is its number. Then, output the quantities of water that are measurable, in ascending order, separated by single spaces. A quantity of water is measurable if Joseph can perform some (possibly empty) sequence of actions from the initial configuration to obtain that quantity in some test tube.

Sample Input/Output

Sample input `water.in`

```
1
2 4 2 0
```

Corresponding output

```
Data set 1: 0 2 4 6 8
```