NEW ZEALAND PROGRAMMING CONTEST

2012

PROBLEM SET

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.

- Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar.

- Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected! You have been warned.

- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. spaces or tabs.

- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.

- Unless otherwise stated, a *word* or a *name* is a continuous sequence of letters.

- Unless otherwise stated, a *string* is a continuous sequence of visible characters.

- Unless otherwise stated, words and names will contain no more than 60 characters, and strings will contain no more than 250 characters.

- If it is stated that ‘a line contains no more than *n* characters’, this does not include the character(s) specifying the end of line.

- Input files are often terminated by a ‘sentinel’ line, followed by an end of file marker. This line should not be processed.

Please also note that the filenames of your submitted programs may need to follow a particular naming convention, as specified by the contest organisers at your site.

PROBLEM A **LEAP YEARS** **3 POINTS**

You are probably aware that 2012 is a leap year. We know this because 2012 is exactly divisible by 4 – and we have an Olympic Games!

Since the modern games started in 1896, the summer games have only once been held in a non leap year, the 1900 games in Paris. This is because 1900 is exactly divisible by 100, so it was not a leap year. What about the 2000 games in Sydney? 2000 was a leap year because 2000 is exactly divisible by 400.

For this problem you will be given a list of years and have to decide which are leap years and which are common years (ie not leap years).

Input consists of a series of lines, each line containing a single number Y (1582 <= Y <= 9999) which represents a year. The last line will contain 0 – this year should not be processed.

Output consists of one line for each line of input. The line will consist of the word `leap` if the year is a leap year or `common` if it is not.

Sample Input	**Output for Sample Input**
`2012`	`leap`
`2011`	`common`
`1995`	`common`
`0`	

PROBLEM B **TRIANGLES** **3 POINTS**

We know that triangles have 3 sides. Triangles may be classified according to the lengths of those sides, as shown in this table:

Classification	Description
Equilateral	All three sides are of equal length.
Isoscoles	Two sides are of equal length, the other side is different.
Scalene	All three sides have different lengths.

There is one other possibility. It is possible to give three lengths of sides which would not form a valid traingle. For example 6, 3, 2. If the sum of the lengths of the two shortest sides is not greater than the length of the longest side, then the sides do not represent a triangle – the data is invalid.

In this problem you will be given values for the three lengths of the sides of a triangle and asked to classify it.

Input

Input consists of a number of lines, each line containing three positive integers, not more than 1,000, separated by single spaces. The last line of input will be `0 0 0` – do not process this line.

Output

For each line of input, produce one line of output containing a single word (Equilateral, Isoscoles, Scalene or Invalid) which is the classification of the lengths shown in the input.

Sample Input

```
7 7 7

6 5 4

3 2 5

6 2 6

0 0 0
```

Output for Sample Input

```
Equilateral

Scalene

Invalid

Isoscoles
```

PROBLEM C **WHEN DO WE FINISH?** **3 POINTS**

In this problem you are given the starting time of an event and its duration. All you have to do is to say when the event ends.

Input

Input consists of a number of lines, each line representing an event. The last line of input will be `00:00 00:00` – do not process this line. Each event is represented by two times separated by a space. Each time is in the format `hh:mm`. The first time is the start time of the event, the second time is its duration. For the start time, a legal time using a 24 hour clock will be supplied. For the duration, the minutes will be from 0 to 59 inclusive, but the hours may be from 0 to 96 inclusive.

Output

For each line of input, produce one line of output containing a single time, also in the format `hh:mm`, being the end time of the event as a legal 24 hour time. If this time is not on the same day as the start time, the time will be in the format `hh:mm +n`, where n is the number of days after the starting day.

Sample Input

```
03:50 06:10

15:25 26:30

23:59 00:01

00:00 00:00
```

Output for Sample Input	Explanation
`10:00`	Same day so no + needed.
`17:55 +1`	The day after the start day.
`00:00 +1`	Midnight – which is the following day.

PROBLEM D **CAR PARK** **3 POINTS**

Our local car park needs your help. They have a fixed number of spaces for parking and want to be able to tell drivers who are approaching the park just how many spaces are available to use. They also need to stop the entry barrier from opening if the park is full. If your system works, they will sell it to other car park owners.

Input

Input consits of a number of scenarios, each one representing a different car park. The first line consists of two integers, `S` and `C` (10 <= S <= 500, 0 <= C <= S). `S` is the number of spaces in the car park overall, and `C` is the number of cars currently parked there. The last scenario contains `0 0` – this scenario should not be processed.

The second line of each scenario consists of a string of up to 255 characters, each character being one of the upper case letters `I` or `O`. This string represents a stream of data from the entry and exit barriers. `I` means that a car has attempted to come in to the car park, `O` that a car has driven out. If the car park is not full, a car attempting to enter will be allowed in and counted. If all the spaces in the car park are filled with cars, a car attempting to enter will be refused entry and not counted. (Such a car may try again later).

If there are no cars in the car park, an `O` in the data stream at that point represents an error, and processing of that scenario should stop.

Output

Output consists of a single line for each scenario. If there has been an error in the data stream, it will contain just the word `error`. Otherwise, it will be a single integer representing the number of cars in the car park at the end of the scenario.

Sample Input

```
50 12

IIOIOOIOIIIOOIOIOOII

25 3

IOOIOOOOIIO

0 0
```

Output for Sample Input

```
14

error
```

PROBLEM E **WEB PAGES** **10 POINTS**

You probably know about HTML, the mark up language used to create Web pages. HTML code contains a number of tags which are expected to follow certain rules.

In this problem we will be concerned with two of these rules:

1 Every opening tag has to have a corresponding closing tag

2 All tags must be properly nested.

Tags are marked by angled brackets which contain a keyword, such as `<body>` or `<strong>`. These are opening tags, the corresponding closing tags having `/` before the keyword, ie `</body>` and `</strong>`. It is possible for a tag to be both opening and closing, such as `<br />`, which complies with rule 1.

To be properly nested, if a tag is opened inside another tag, it must be closed before the other tag closes. For example

`<body> <strong> </strong> </body>` is properly nested

`<body> <strong> </body> </strong>` is not, and breaks rule 2.

Notes

If there are no tags present, the text complies with both rules.

Attributes may be present within an opening tag, such as

```
<a href="http://www.nzprogcontest.org.nz">This is a link</a>
```

The closing tag has only to match the keyword, not the attributes.

Input

Input will consist of a number of lines of HTML code, each line containing from 0 to 255 characters. The last line will contain a single # character – do not process this line. Within the text of each line will be zero or more tags. No angle bracket will be present unless it is part of a properly formed tag.

Determine whether or not the HTML meets the rules specified above.

Output

Output will consist of a single line for each line of input. The line will contain either the word `legal`, or the word `illegal`.

[Turn over for sample data]

Sample Input

```
<body> <strong>Oops, this is</body> naughty </strong>
<body> <strong>Hello</strong> <br /> </body>
Just text, no tags.
<p> Oh dear, we are missing something.
<a href="http://www.nzprogcontest.org.nz">This is a link</a>
#
```

Output for Sample Input

```
illegal
legal
legal
illegal
legal
```

PROBLEM F **MAPS** **10 POINTS**

You are surveying photographic images of Antarctica for the UNADID, looking for evidence of secret (and illegal) oil drilling rigs. You are to report how many you discover.

You will be given a number of scenarios. Each scenario consists of an image of a secret rig and a map of a section of snowfield. Snow is represented by a dot (.) and part of a building by a X. You are to count the number of times an exact replica of the rig in the image is found on the map. "Exact" here means you do not have to count rotations or mirror images.

Input

The first line of input will be a single integer containing the number of scenarios.

Each scenario will start with one line containing two positive integers Li and Ci representing the number of lines and the number of columns in the image of the rig. Both numbers will be in the range 1 to 12 inclusive.

Li lines then follow, each line containing Ci characters. Each character will be an upper case X or a dot. These lines represent the image of the secret rig, which has been trimmed so there are no unnecessary snow (.) elements surrounding the rig (X) elements.

The next line will contain two further integers, Lm and Cm, representing the lines and columns of the map to be searched. Both numbers will be in the range 1 to 32, with Lm being no less than Li and Cm being no less than Ci.

Lm lines then follow, each line containing Cm characters. Each character will be an upper case X or a dot. These lines represent the map you are to search.

Output

Output consists of a single integer (on a line of its own) for each scenario. The integer is the number of times a secret rig was counted in the map.

[Turn over for sample input and output]

Sample Input

```
1

2 3

XXX

.X.

5 16

...XXX...X.X.XXX

X...X...XX.X.XX.

....X.X.........

X....X....XXX...

....XXX....X....
```

Output for Sample Input

```
2
```

Explanation

The two counted rigs on the map are highlighted below on the left hand diagram. The two buildings highlighted in the right hand diagram do NOT count. The top one has an extra building element (bottom left) while the bottom one is upside down.

PROBLEM G **SHIRTS** **10 POINTS**

Jane and Tim are getting married. They are so compatible because they keep their shirts in their wardrobes in the same colour and size order. Jane has bought her own house so Tim will be moving in after they are married. His shirts will need to be merged in with hers, sorted by size then colour.

If you do a good job, they will make your solution available to other newly weds that are known to have similar habits, so you will be given more than one set of data to test.

Input

Input consists of a number of scenarios. A scenario begins with a single number, W (0 < W <= 30) on a line of its own being the number of shirts belonging to the wife. If this number is 0, then it signals the end of input.

The number is followed by W lines, each line being two upper case letters. The letters represent the size and colour of a shirt. Sizes are S, M and L. Colours are B for Blue, K for blacK, N for browN, O for Orange, P for Purple, R for Red and W for White.

The next line is another single number, H (0 < H <= 30) on a line of its own being the number of shirts belonging to the husband.

The number is followed by H lines to represent the husband's shirts using the same two letter codes as for the wife.

Output

Output consists of a single line for each scenario. The line contains all of the shirts from the input, separated by single spaces. The shirts are sorted firstly by size (small before medium before large) then by colour (alphabetical order of the representative letter).

Sample Input

```
5
SB
SB
SP
MP
MR
6
MB
MK
MP
LK
LN
LW
0
```

Output for Sample Input

```
SB SB SP MB MK MP MP MR LK LN LW
```

PROBLEM H **DISCOUNTS** **10 POINTS**

Peter's business is flagging so he has been looking for a way to improve sales. His latest idea is a “buy three get one free” type scheme. He is hoping that special offers like this will encourage customers into his shop.

Your job is to write a program to help Peter by making it easier for customers to know how much they can save.

Input

Input consists of a number of product scenarios. A scenario begins with the name of the product on a line of its own. The name consists of 1 or more words which will be separated by spaces. The final line contains a product name consisting of a single # symbol; do not process this product.

The second line of each product scenario consists of two integers, PD and PC (0 <= PD <= 50, 0 <= PC <= 99), separated by a space, which is the price per item of the product in dollars and cents respectively. PD and PC will not both be zero.

The third line of each product scenario consists of an integer, D (0 < D <= 10) on a line of its own being the number of deals available for this product.

The number is followed by D lines, each line being two integers B and F. (0 < B F <= 100) separated by a space. B is the number of products that must be bought and F is the number available free if this number is bought.

The next line is another single number, E (0 < E <= 30) on a line of its own being the number of examples to follow.

The number is followed by E lines each containing a single positive integer less than 500. Each number represents a quantity of items to be purchased. You have to use the available deals to work out the best saving the customer can make. Note that in calculating the best deal, a customer does not have to take all the free items offered.

Output

Output consists of a section for each product. The section starts with the name of the product on a line of its own. This line is followed by E lines, one for each example in the input. Each line will be of the format

`Buy N, save $D`

N is the number of items bought and D is the amount saved compared to having no free items. The amount D will be in the format

`d.dd`

That is there will be at least one digit for the dollars, a decimal point and two digits for the cents.

A blank line separates each product section.

[Turn over for sample input and output]

Sample Input

```
Baked Beans
0 95
3
12 1
36 5
100 12
6
10
26
40
41
54
153
#
```

Output for Sample Input

```
Baked Beans
Buy 10, save $0.00
Buy 26, save $1.90
Buy 40, save $3.80
Buy 41, save $4.75
Buy 54, save $5.70
Buy 153, save $16.15
```

Explanation

Must buy 12 for any saving.
Save on 1 item after buying 12, repeat for another 12.
Must pay for 36, get 4 free (of the 5 available)
Must pay for 36, so get 5 free.
Pay for 36, get 5 free, pay for 12 more get 1 more free.
Pay for 100, get 12 free, pay for 36 more get 5 more free.

Problem J Microspikes 30 Points

In a study of domestic power consumption, researchers built a simulator for New Zealand homes. For each home the software simulates the power consumption of appliances. The goal of the project was to identify microspikes in power usage – short periods of time during which total power consumption rose above a specified limit.

Before a simulation starts all appliances are turned off (using no power). At various times during the simulation period appliances will increase or decrease their power usage. Every time this happens the simulator outputs a record with the appliance number, the time (seconds) since that appliance's last change (or since the start of the simulation), and the change in power level (watts). For a given appliance, records are in order, but unfortunately the simulation was written in such a way that results for different appliances are randomly interleaved. In particular it cannot be assumed that a record for appliance A, written before a record for appliance B, reports an event on appliance A that occurred before B's event.

Your task is to write a program to read files of appliance records, and count the number of microspikes that occur. For any given simulation you will be given a power threshold M and a time threshold S. A microspike occurs if the power level P satisfies P > M for a period of time T: 1 <= T <= S.

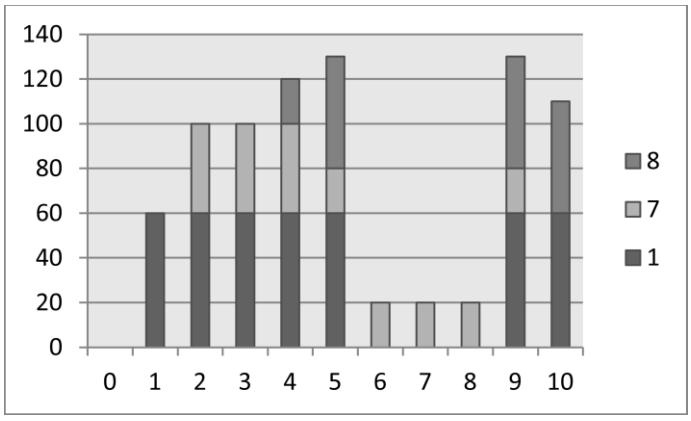
Input Specification

Your input is data from a number of simulations. The data for each simulation begins with a line holding three integers (T, M and S), separated by single spaces. T is the total simulation time in seconds (0 <= T <= 100,000); M is the power threshold (0 <= M <= 10^9); and S is the time threshold (1 <= S <= 1000). A line with three zeroes ends the input. Following are N power change lines (0 <= N <= 1,000,000). Each holds three integers (a, t and p), again separated by single spaces: 'a' is the appliance number (1 <= a <= 100,000); 't' is the time in seconds since that appliance's last change (0 <= t <= T); and 'p' is the change in power level (-10,000 <= p <= 10,000). A line with three zeros indicates the end of data for that simulation.

You can assume that the power level of an appliance never goes negative, and that total power consumption never exceeds 1,000,000,000. A microspike is only counted if both increase from below to above the threshold and reduction to below or at the threshold are reported (At the end of simulation appliances may not have been turned off).

Output Specification

For each simulation one line of output is required, with the number of microspikes observed.



Total power consumption for sample input. Labels on the horizontal axis are the times for the starts of the seconds represented by the bar. All power changes occur instantaneously at the start of a second. Note that the column at time 10 is not part of the simulation. It is included in the chart to show the power level that continues indefinitely after the simulation end.

Sample Input (see graph)

```
10 100 2
8 4 20
8 1 30
1 1 60
7 2 40
7 3 -20
1 5 -60
1 3 60
7 5 -20
8 1 -50
8 3 50
0 0 0
0 0 0
```

Sample Output

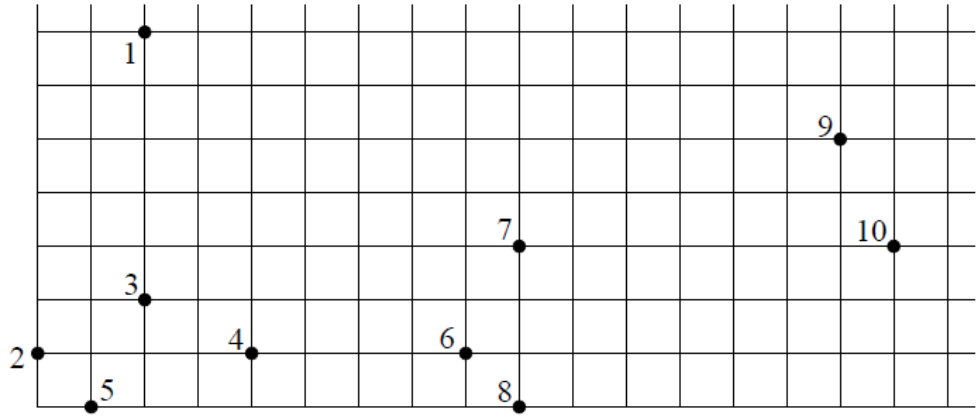
```
1
```

Problem K Constellations 30 Points

One night, while camping out in the wide open spaces, Big Ed was looking at the stars. Now Ed never bothered to learn his constellations, but decided that grouping the stars was a reasonable thing to do after all. But Big Ed was going to do it in a sensible manner. He decided on the following simple rules:

- Every star is in the same constellation as its closest neighbor.
- If the closest neighbour is not unique, then the star and all its closest neighbours are in the same constellation.
- If *A* is in the same constellation as *B*, then *B* is in the same constellation as *A*.
- If *A* is in the same constellation as *B*, and *B* is in the same constellation as *C*, then *A* is in the same constellation as *C*.

For example, if the picture of the sky looked like the following:



Then there are 3 constellations: {1, 2, 3, 4, 5}, {6, 7, 8}, {9, 10}.

Input

Input will consist of a sequence of sky descriptions. Each begins with a single integer $n: 0 < n \leq 500$ on a line, indicating the number of stars in the universe. The coordinates for the *n* stars follow as a pair $x, y: 0 \leq x \leq 1000$. A value of $n = 0$ indicates end of input.

Output

For each sky description, print a single line of the form

`Sky` *s* `contains` *c* `constellations.`

where *s* is the number of the sky description (starting at 1) and *c* is the number of constellations.

Sample Input

```
10
0 1
16 3
1 0
2 7
9 0
4 1
2 2
8 1
9 3
15 5
5
10 10
10 11
20 10
20 11
15 5
0
```

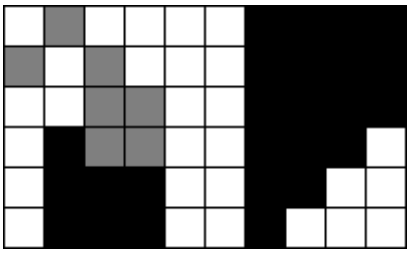
Sample Output

```
Sky 1 contains 3 constellations.
Sky 2 contains 1 constellations.
```

Problem L Image Segmentation 30 Points

You have been recruited to the staff of a medical imaging company. Your task is to build a test program as part of an image analysis system. Many devices produce pictures of structures inside the human body, like x-ray systems and ultrasonic scanners. A first step in automatic analysis of such images is to separate the image into connected areas, of like appearance or colour.

The way in which we define connected is most easily explained with a small artificial example. Consider the magnified 6 row by 10 column pixel image on the left below. There are three colours of pixel, white, grey and black. We consider two pixels of the same colour to be part of the same area if they are adjacent, horizontally, vertically or diagonally. So the set of 7 grey pixels form one connected area. In all there are 5 connected areas in this image. The areas are shown in the image to the right, where pixels in the same area have the same number.



1	2	1	1	1	1	3	3	3	3
2	1	2	1	1	1	3	3	3	3
1	1	2	2	1	1	3	3	3	3
1	4	2	2	1	1	3	3	3	5
1	4	4	4	1	1	3	3	5	5
1	4	4	4	1	1	3	5	5	5

Your task is to write a program which will examine an image and report the number of areas. There are some extra details to consider.

- Your image will be coloured. The pixel colour is recorded as a three vector (R, G, B) where values R, G, and B are integers in the range 0 to 255 inclusive.
- It turns out that real images rarely have regions of exactly one colour. Instead you will have to treat similar colours as though they were the same. This will be done by 'banding'. With each image, you will be given a 'band size' S. This must be used to convert each colour value into a band number. For example: with S = 32, Red, Green or Blue values in the range 0..31(incl) will be taken as band 0, 32..63 will be band 1, etc. The colour (5, 32, 76) will be converted to a band number version (0, 1, 2). Two colours should be considered to be the same if they have the same band number form.
- For use in medical work, small areas can be ignored. For each image you will be given a size limit value L. You should not count areas that include less than L pixels.

INPUT

Input takes the form of a sequence of images. The data for each image starts with a line holding 4 integer values, separated by spaces: H W S L being image height, image width, band size and area size limit. W and H lie in the range 1 to 1024 (inclusive). This will be followed by H lines of data. Each line of data holds W triples, being the R, G and B values for successive pixels. Numbers are separated by single spaces. The end of input is denoted by a line with 4 zeroes.

OUTPUT

For each image, output a single line with the number of areas in the image (not counting those that are too small).

SAMPLE INPUT

```
6 10 100 7
0 0 0 128 128 128 0 0 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255
128 128 128 0 0 0 128 128 128 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255
0 0 0 0 0 0 128 128 128 128 128 128 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 255 255 255
0 0 0 255 255 255 128 128 128 128 128 128 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 255 255 255 0 0 0 0 0 0
0 0 0 255 255 255 255 255 255 255 255 255 0 0 0 0 0 0 255 255 255 0 0 0 0 0 0 0 0 0
0 0 0 0
```

SAMPLE OUTPUT

```
4
```

Problem M Team Dessert 100 Points

One of the problems of dieting is that it can be a lonely and uninteresting experience. To solve this problem a new company Olympic Slimmers Inc (OSI for short) is trying to develop dieting as a team sport. Their first game is called the Team Dessert challenge. In this game a line of dessert dishes, each labelled with its weight, are placed on a long table. Players are organised into two teams. Players take turns at selecting their dessert , alternating between teams. Ie: For teams Red and Blue, if a player from Blue starts, the next to play is from the Red team, then a player from Blue, then Red, etc. There is one important restriction on selection. When a person makes a choice they must take a dessert from one of the ends of the line – this means that each choice is just between two options (unless they choose last, in which case they have no choice).

- Teams are as close to equal in number of players as possible.
  If there is an imbalance then the team with more people starts first.
- Everyone must select a dessert.
- The number of desserts on the table is equal to the number of players.

The team with the smallest total weight of dessert wins.

The game has proved popular, so teams can be quite large.

Your task is to write a program which, given a list of the dessert weights in order of placement, can select the best (might not be winning) strategy for a team that starts first.

INPUT

Input takes the form of a sequence of sequence of problems. For each problem the first line holds one number – N, the number of people (1 <= N <= 1000). End of input is signalled by N being zero. The next lines hold the weights of desserts in order (each weight W: 1 <= W <= 100). There will be some number of values per line (at least one) such that the line length is never greater than 80 characters. The numbers are separated, may be preceded and may be followed by one or more spaces.

OUTPUT

For each problem in the input, you must calculate the size of each team and then calculate the minimum weight that the team making the first selection can be sure of achieving (assuming that their opponents play the best strategy). Output this as a line with a single number.

SAMPLE INPUT

```
4
10 10 9 10
5
10 10 10
 10 10
4
10 4 1 10
0
```

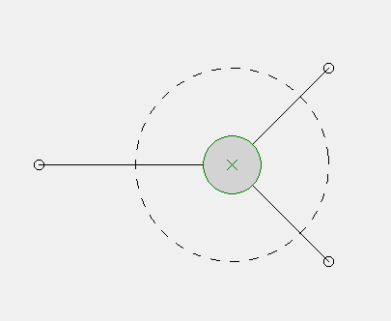
SAMPLE OUTPUT

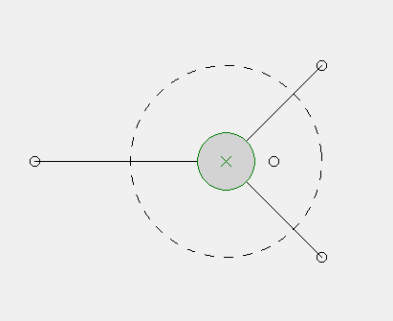
```
19
30
11
```

Problem N Bat Positioning 100 Points

You have been asked to help with the programming in a cooperative network game. The idea is that players cooperate in moving a bat (disk shape) around on a playing surface. When the whole game is finished there will be bouncing balls that can be hit with the bat when it is in a suitable position. For the moment we are just programming the bat movement. The game is played on a 1000 by 1000 unit board. Each player can position and move a pointer on the board. The pointer is connected by an elastic cord to the centre of the bat disk. The bat will move about as the pointer locations change, coming to rest in an equilibrium position, where the forces acting on it balance. Your task it to write a programme that takes pointer positions as input and outputs the position of the bat.

For example, as shown in the diagram there are pointers at (300, 400), (600, 300) and (600, 500). Each applies a force proportional to its distance from the centre of the bat, resulting in an equilibrium position of (500, 400) for the bat.





A complicating issue is that elastic cords have a minimum length, below which they cannot exert any force. In our system this minimum length is 100 units. Pointers closer than 100 units from the centre of the disk exert no force. In the image to the right a new point has been added at (550, 400). It is shown without a connecting line as it is exerting no force. (The dashed circle in the images shows the 100 unit limit.)

Fortunately a game world can violate proper physics. For this game the force applied by a pointer is proportional to its distance from the disk centre. The result can be some numerical instability when pointers are close to the 100 unit limit. For the purposes of this programming problem, you can assume that pointers do not finish close to the limit.

Input

Input will consist of a sequence of game situations. Each begins with a line holding a single integer $n: 0 \leq n \leq 30$. indicating the number of pointers. It is followed by one line for each pointer, holding its coordinates $x, y: 0 \leq x, y \leq 1000$ as two integers separated by a space. Input is terminated by a zero value of *n*.

Output

For each sky game situation, output one line with the equilibrium position of the centre of the disk, as shown in sample output. Coordinates should be rounded to the nearest integer value.

Sample Input

```
4
300 400
600 300
600 500
550 400
0
```

Sample Output

```
(500,400)
```

Problem O Gooseberry Tart BASIC 100 Points

Many fondly remember the Commodore 64 personal computer and its implementation of the BASIC programming language. Recently there has been a surge of nostalgic development effort – as teams of engineers and programmers compete to create modern versions of this iconic system. You have been asked to support team 'Gooseberry Tart' by building them a BASIC interpreter (GTB1). Of course now that computers have more memory and are expected to process video streams in real time, it is vital that your system run fast. Your task is to build a super fast interpreter for a limited subset of the language as specified below.

The GTB1 Language

- There is only one type of variable – the 32 bit signed integer. All <expression>s are integer valued. There is a comparison (Boolean) syntax in the IF statement.
- Variable names start with a letter (A .. Z, a..z) and may continue with letters or digits.
- Variable names can be of any length, but only the first two characters matter. Names are not case sensitive. For example the variables Fred, Fr, Freda and fRE are all the same.
- Reserved words are not case sensitive. No variable name may have the same first two characters as any reserved word. Only the first two characters of a reserved word are required. The reserved words are: LET, GOTO, IF, FOR, TO, NEXT, OUT and COMMENT
- Arithmetic expressions allow () brackets and binary operators + - * / and % (addition, subtraction, multiplication, integer division and modulo) with the usual precedence.
- Arithmetic expressions allow unary minus at the start of an expression or bracketed sub-expression. It has the same precedence as binary + and -.
- A programme consists of a series of instruction lines.
- Each instruction line begins with a number. Line numbers are always in ascending sequence in a programme. A line number must be in the range 1 to 10000 inclusive.
- Each instruction line can be: an Assignment statement; a GOTO statement; an IF statement; a FOR statement, a NEXT statement, an OUT statement or a comment

The Assignment statement takes the form: LET <variable> = <expression>

The GOTO statement takes the form: GOTO <line number>

The IF statement takes the form: IF <expression> <comparison> <expression> GOTO <line number> where <comparison> can be =, <, >, <=, >= or <> (equal, less than, greater than, less than or equal to, greater than or equal to and not equal.

A looping construct requires matching (same <variable>) FOR and NEXT statements. It starts with the FOR statement, follows with some statements to form the body of the loop, and ends with the NEXT statement. It is possible to have nested FOR/NEXT structures. Such nesting must be proper in that the inner FOR NEXT structure must be between the FOR and NEXT instructions of the outer.

A FOR statement takes the form: FOR <variable> = <start expression> TO <end expression>

A NEXT statement takes the form: NEXT <variable>

Loop execution begins by assigning the <start expression> to the <variable>. The body of the loop executes at least once. The NEXT statement adds 1 to the value in the variable. If the value is less than or equal to the <end expression> the body of the loop executes again. Note that <end expression> is evaluated on every occurrence of the NEXT statement.

NOTE: It is permissible to GOTO into or out of a FOR/NEXT structure.

Statements are normally executed in numerical order (ie: the sequence in which they are written), starting with the first line of the programme. This order can be altered by GOTO or IF statements. A programme finishes when it runs off the end.

Variables need not be declared. They come into existence when first assigned or referenced. If a variable is referenced before it is assigned, it will have the value zero.

A COMMENT statement takes the form: COMMENT <any text>

An OUT statement takes the form: OUT <expression>

The OUT statement outputs one line of output, being the value of the expression.

INPUT

Input takes the form of a sequence of programmes. Each programme starts with a number *N*, being the number of lines in the programme: $1 \leq N \leq 1000$. This is followed by *N* lines of statements. No statement is longer than 80 characters. Tokens may be separated by more than one space. There may be spaces at the start or end of lines. There are no tabs. There need be no spaces around operators or parentheses in an expression. Each line has a line number and a statement. End of input is signalled by *N* being zero. You may assume that all programmes are syntactically correct and will terminate (in reasonable time if your interpreter runs fast).

OUTPUT

For each programme output "Programme <i>" (where i is 1, 2, 3, etc.), followed by any lines of output generated by the execution of the programme.

Note: Test data may require execution of up to 10^9 (10 to the power 9) instructions.

SAMPLE INPUT

```
1
1000 OUT 225
6
10 OUT 1
20 LET S = 0
30 FOR I = 1 TO 100
40 LET S = S + I
50 NEXT I
60 OUT S
0
```

SAMPLE OUTPUT

```
Programme 1
225
Programme 2
1
5050
```