Computer Science Competition
Region 2021
Programming Problem Set

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

5. A penalty of 5 points will be assessed each time that an incorrect solution is submitted. This penalty will only be assessed if a solution is ultimately judged as correct.

II. Names of Problems

Number	Name
Problem 1	Adéla
Problem 2	Bruce
Problem 3	Esteban
Problem 4	Guozhi
Problem 5	Kevin
Problem 6	Mahesh
Problem 7	Maximo
Problem 8	Olivia
Problem 9	Randy
Problem 10	Sergey
Problem 11	Terry
Problem 12	Yana

1. Adéla

Program Name: Adela.java Input File: none

Adela needs to print a sideways pyramid. Please write a program that will do that for her. Thanks!

Input: There is no input for this problem.

Output: The exact pattern of asterisks shown in the sample output below.

Sample input: None

Sample output:

```
*
**
***
****
*****
******
*******
********
*********
********
*******
******
*****
****
***
**
*
```

2. Bruce

Program Name: Bruce.java **Input File: bruce.dat**

Bruce has a fencing company. His latest job involves fencing several square plots of land. Oddly enough, all Bruce knows about each plot he is expected to fence, is the area. Of course, to determine the amount of fencing he must purchase for the job Bruce needs to calculate the perimeter of each plot. Write a program that will read the area of a square plot of land and then print the amount of fencing he must purchase.

Input: A single value N (N <= 10) representing the number of plots to be fenced. The next N lines will each contain a single whole number A (1 <= A <= 40000) representing the area of each of the plots of land measured in square feet.

Output: N lines where each line contains the number of linear feet required to fence the plot. Each value should be rounded to four decimal places.

Sample input:

```
4
100
50
85
20000
```

Sample output:

```
40.0000
28.2843
36.8782
565.6854
```

3. Esteban

Program Name: Esteban.java **Input File: esteban.dat**

Esteban hates hot humid weather! He has visited many places across Texas and observed the western and northern areas of Texas seeming cooler than the central and eastern areas even though actual reported temperatures may be about the same or even higher. He found a heat index formula on the web but it is way too complex to use a calculator for multiple samples. He also found formulas to convert between °F and °C, because his grandparents live in Spain, which he visits every summer, and they report temperatures in Celsius. The heat index (**HI**) formula shown below uses ambient (dry shade) air temperature in °F (**t**) and relative humidity % (**h**) to produce a heat index, often reported as the "feels like" temperature.

$$\begin{aligned}\mathbf{HI} &= -42.379 + 2.04901523\mathbf{t} + 10.14333127\mathbf{h} - 0.22475541\mathbf{th} \\ &\quad - 6.83783\cdot10^{-3}\mathbf{t}^2 - 5.481717\cdot10^{-2}\mathbf{h}^2 + 1.22874\cdot10^{-3}\mathbf{t}^2\mathbf{h} \\ &\quad + 8.5282\cdot10^{-4}\mathbf{t}\mathbf{h}^2 - 1.99\cdot10^{-6}\mathbf{t}^2\mathbf{h}^2\end{aligned}$$

$$C = \frac{5}{9}(F - 32) \quad \text{and} \quad F = \frac{9}{5}C + 32$$

However, there are limitations with the **HI** formula. To start with, the formula is valid only when temperature is at or above 80 °F and there are two situations that require an adjustment to the calculated heat index:

1. When humidity is less than 13%, the following adjustment is subtracted from **HI**:
   ADJUSTMENT = ((13 – **h**) / 4) * SQRT((17 – ABS(**t** – 95.0)) / 17)
   where ABS and SQRT are absolute value and square root functions, respectively.

2. When humidity is greater than 85%, the following adjustment is added to **HI**:
   ADJUSTMENT = ((**h** – 85) / 10) * ((87 – **t**) / 5)

In addition, there are two situations when the above formulas produce invalid results:

1. When the calculated heat index is over 140 °F
2. When the calculated heat index is below ambient air temperature (°F) with humidity at or above 40%

Input: First line contains a single whole number, N ≤ 50, number of test cases that follow. Each test case consists of a single line with whitespace separated values: either 'F' or 'C' to indicate which temperature is provided, ambient air temperature, and relative humidity. Both numeric inputs will be whole numbers with 0 ≤ temp ≤ 150 and 0 ≤ humidity ≤ 100.

Output: The case # followed by a colon and the heat index with 1 digit following the decimal point, matching the input temperature type with either 'F' or 'C' as shown below. When the calculated heat index is invalid, add a space and "INVALID" to the output line.

Sample input:

```
7
F 97 44
C 42 47
F 79 12
C 26 52
F 90 11
F 80 40
C 35 86
```

Sample output:

```
#1:105.6F
#2:59.1C
#3:77.5F INVALID
#4:26.6C INVALID
#5:85.5F
#6:79.9F INVALID
#7:60.6C INVALID
```

4. Guozhi

Program Name: Guozhi.java **Input File: guozhi.dat**

After the heist *(as described in Terry's story later in the packet)* Terry passes the stolen contraband to Guozhi to be smuggled out of the city. He needs some help from you and your team to accomplish this dangerous task. Can you help Guozhi find a way to the safe house without getting caught?

The city can be modeled as a grid with R rows and C columns. Some cells in the grid have trees, and others have cameras. Guozhi cannot enter a cell with a tree or camera.

During each time-unit, the cameras rotate clockwise by 90 degrees. That is, if the camera was looking in a certain direction in one time-unit, it would be looking to the right of that direction in the next time-unit. Cameras can see every cell in a straight line from the camera in the direction it is facing but cannot see through trees or the safe house. Cameras can see through other cameras in their current line of sight. Fortunately, the team's tech guy has provided Guozhi with a special computing device that can sense the current state of each camera so he can safely contemplate his next move.

At the start of each time-unit T, if there is a camera that can see Guozhi, he gets caught. Otherwise, the cameras all rotate 90 degrees clockwise each time-unit, and Guozhi can opt to either stay still or move to an adjacent empty cell. Two cells are adjacent if they have a side in common. How fast can Guozhi get to the safe house?

DISCLAIMER: The University Interscholastic League does not endorse using computing skills for breaking the law or other evil misdeeds.

Input: The first line of input is an integer T ($1 \leq T \leq 20$), the number of test cases. Each test case begins with two integers R and C ($1 \leq R * C \leq 1{,}000$), the number of rows and number of columns. The next R lines each have C characters, which are one of the following 8 characters:

`G` - the starting location of Guozhi
`H` - the location of the safe-house
`E` - A camera which is facing east at time T = 0
`N` - A camera which is facing north at time T = 0
`S` - A camera which is facing south at time T = 0
`W` - A camera which is facing west at time T = 0
`.` - an empty square
`#` - a tree.

Output: For each test case, if it's impossible to escape, output -1. Otherwise, output the minimum amount of time to get to the safe house. Format your output with the case number as in the samples.

Samples and explanation on next page.

Guozhi sample input:

```
3
3 3
..G
N..
..H
3 3
..H
N#.
..G
4 5
.###H
S....
.G###
#..N#
```

Sample output:

```
Case #1: 3
Case #2: 2
Case #3: -1
```

Sample explanation:

In the first sample, Guozhi cannot immediately move south because the camera would catch him at the beginning of time T = 1. The optimal strategy is to wait in the starting cell for 1 time-unit, and then move south twice. This takes 3 total time units.

In the second sample, a tree blocks the camera so Guozhi can just move north twice.

In the third sample, there is no way to get to the exit without being seen by the leftmost camera.

5. Kevin

Program Name: Kevin.java **Input File: kevin.dat**

Kevin is an Irish exchange student. He is a serious runner and keeps a log to track his performance, recording each run with the date, the distance in kilometers, and the time in hours, minutes and seconds. He has been learning how to use a spreadsheet to generate some useful summary information but is not very confident in his results. He has asked your UIL programming team to write a program to verify his spreadsheet results.

For each log entry, Kevin computes the average pace as minutes per kilometers which is just time divided by distance. He tabulates the overall total distance, total time spent running, and the daily averages of both. He also keeps up with his overall longest distance and time, and his run with overall fastest pace in minutes per kilometer. He wants to also compute the average distance and pace for all short runs (5.0 km ≤ distance < 10 km) and the same for all medium runs (10 km ≤ distance < half a marathon) and for all long runs (≥ half a marathon). He has not yet run a full marathon which is 42.195 km but has plans to do so in the future.

Kevin would also like to obtain some statistics for his best 7-day running period, that is, **seven consecutive days of running with a total distance greater than any other seven consecutive day period.** He wants the total distance and time, average distance and time, and average pace across 7 consecutive days. He wants to know the first and last date of that 7-day period.

Input: A list of log entries of unknown quantity, but fewer than 500. Each line will contain a date in the format m/d, a distance in kilometers which will be < 50.00, and a time in the format h:m:s. The items will be separated by whitespace. There is only one entry for each day he runs, but he does not run every day. He makes no entries for non-running days. Log entries are in chronological order from oldest to newest. Assume February dates are always in a leap year, like the year 2020, and dates may wrap from one year to next. There will be only one February in the test data. A 7-day period with consecutive days of running will exist in each test data set. Test data will exist for short, medium, and long categories.

Output: Overall total distance, overall total time, and daily averages for both. Longest distance, longest time, and fastest pace of all runs. Average distance and pace for short, medium, and long runs. Start and stop dates of best-distance 7-day stretch with totals and averages for both distance and time and the average pace for that period. Format all items as shown in sample output. Single-digit seconds must always have leading 0 and, when hours are displayed, single-digit minutes must have a leading 0. Results must be worded and formatted as shown in sample.

Sample input and output are on next page…

Kevin Sample input:

```
2/2             9.74        0:29:37
2/3             24.93       1:32:14
2/4             24.45       1:08:13
2/5             8.58        0:24:48
2/6             7.66        0:28:11
2/7             14.4        0:47:57
2/8             11.04       0:31:21
2/10            29.88       1:19:47
2/11            5.39        0:19:47
2/12            8.25        0:23:56
2/13            6.75        0:23:50
2/14            20.73       1:10:04
2/14            22.06       1:08:36
2/15            5.63        0:17:20
2/16            20.48       1:07:23
2/17            24.32       1:26:49
2/18            15.56       0:52:17
2/19            27.98       1:25:20
2/20            26.74       1:11:08
2/22            5.74        0:16:39
2/23            13.47       0:39:04
2/24            16.42       0:45:49
2/25            7.57        0:26:39
2/26            16.78       0:45:18
2/27            13.00       0:36:16
2/28            22.06       1:07:43
2/29            6.81        0:25:04
3/1             7.66        0:27:02
3/2             26.95       1:32:59
3/5             20.86       0:56:32
3/6             8.39        0:21:54
3/7             29.45       1:50:26
3/8             23.02       1:10:26
3/9             15.32       0:41:31
3/10            25.38       1:11:04
3/11            9.81        0:35:43
3/12            18.11       0:47:16
3/13            26.15       1:07:44
3/14            26.91       1:19:07
3/15            14.49       0:51:26
3/16            5.71        0:16:10
3/17            7.36        0:26:39
3/18            13.05       0:33:56
3/19            13.18       0:37:18
3/20            27.81       1:23:42
```

Sample output:

```
Total distance = 736.03
Total time = 37:42:05
Average distance = 16.36
Average time = 0:50:16
Longest distance = 29.88
Longest time = 1:50:26
Fastest pace = 2:35
Short run average distance = 7.40
Short run average pace = 3:16
Medium run average distance = 15.79
Medium run average pace = 2:58
Long run average distance = 25.87
Long run average pace = 3:05
Best 7-day streak 3/7 - 3/13
Total distance = 147.24
Total time = 7:24:10
Average distance = 21.03
Average time = 1:03:27
Average pace = 3:01
```

6. Mahesh

Program Name: Mahesh.java **Input File: mahesh.dat**

Mahesh is taking an introductory signal processing class for dual credit and has stumbled upon the Hamming Character Code.

The Hamming Character Code is a code in which single-bit errors can be detected and corrected. Each 7-bit ASCII character is embedded in an 11-bit code word called a Hamming character. Bit positions of the hamming character are numbered from most significant to least significant. The encoding is done as follows:

Bit positions:	1	2	3	4	5	6	7	8	9	10	11
Bits:	p	p	d	p	d	d	d	p	d	d	d

Bits 1, 2, 4, and 8, all having bit positions that are a power of two, are **parity bits**. Bits 3, 5, 6, 7, 9, 10, and 11 are the **data bits** which represent the most to least significant bits of the embedded ASCII character. The parity bits are computed so that

bit 1 = (bit 3 + bit 5 + bit 7 + bit 9 + bit 11) mod 2
bit 2 = (bit 3 + bit 6 + bit 7 + bit 10 + bit 11) mod 2
bit 4 = (bit 5 + bit 6 + bit 7) mod 2
bit 8 = (bit 9 + bit 10 + bit 11) mod 2

For example, the ASCII character ‘M’ having 7-bit code 1001101 will be represented by 01110010101.

Suppose the text encoded as a sequence of Hamming characters is transmitted through a channel that can introduce at most one bit error per Hamming character. We may determine the location of the error and correct it as follows. If the Hamming character 01110010101 representing ‘M’ were to be received as 01110010001, the location of the errant bit may be computed as the sum of the positions of the parity bits that disagree with their recomputed values. The offending bit is then complemented to determine the correct Hamming character. Thus the received word 01110010001

∗ bit 1 is 0 1 = (bit 3 + bit 5 + bit 7 + bit 9 + bit 11) mod 2
bit 2 is 1 1 = (bit 3 + bit 6 + bit 7 + bit 10 + bit 11) mod 2
bit 4 is 1 1 = (bit 5 + bit 6 + bit 7) mod 2
∗ bit 8 is 0 1 = (bit 9 + bit 10 + bit 11) mod 2

1 + 8 = 9, which indicates the location of the offending bit! If bit 9 in 01110010001 is changed, it yields the corrected Hamming character for ‘M’, 01110010101. This decoding method works for all Hamming characters with one bit error. Naturally, if there are no bit errors, there will be no discrepancies between parity bits and their recomputed values.

Can you help Mahesh write a program to implement the bit correction described above?

Input: Input starts with a line containing an integer N (1<=N<=150), the number of test cases. Each of the following N lines contains an integer, representing a single Hamming character with at most one bit error. Write a program that decodes the message, correcting for single-bit errors.

Output: The decoded message should be written to the standard output as characters, not integers, all on the same line with no separation.

Sample Input and Output on next page

Mahesh sample input:

```
7
22992
3533
-20667
24407
14937
-17578
23535
```

Sample Output:

```
Hamming
```

Explanation of Sample Output

Input Integer	Input Integer in Hexadecimal	Low-Order 11 Bits	Offending Bit	Resulting Character
22992	59D0	00111010000	5	1001000 = H
3533	0DCD	10111001101	9	1100001 = a
-20667	AF45	11101000101	7	1101101 = m
24407	5F57	11101010111	10	1101101 = m
14937	3A59	01001011001	3	1101001 = i
-17578	BB56	01101010110	0	1101110 = n
23535	5BEF	01111101111	6	1100111 = g

7. Maximo

Program Name: Maximo.java **Input File: maximo.dat**

A set is defined as a collection of well-defined and distinct objects. For example, the set A={1,2,3} has 3 elements, those elements being: 1, 2, and 3. A set A is a subset of another set B if all elements of the set A are elements of the set B. For example, given A={1,2,3} and B={1,2,3,4,5,6}, A is a subset of B, since all the elements of A are also elements of B. The intersection of two sets is defined as the set containing all the elements that are common to both A and B. For example, given A={1,2,3,4,5} and B={3,4,5,6,7}, the intersection of A and B would be the set {3,4,5}. So if A is a subset of B, then the intersection of A and B is simply A. If A is a subset of B and B is a subset of A, then the two sets are equal, and their intersection would be either A or B (since they are the same set).

Maximo needs your help writing a program that given two sets A and B, will determine if A is a subset of B, if B is a subset of A, if A and B are equal, or if neither set is a subset of the other. If neither set is a subset of the other, Maximo would like to know what their intersection is. Can you help him with this?

Input: Input starts with a line containing an integer N (1<=N<=10), the number of test cases. Each test case consists of two lines. The first line will contain a set A, and the second line will contain a set B, both of positive integers between 1 and 100 (inclusive) inside of curly braces. The sets each contain between 1 and 100 distinct elements. Data is comma separated with no spaces.

Output: If sets A and B are equal (meaning A is a subset of B and B is a subset of A), output **"Set A is equal to Set B"**. If set A is a subset of B only, output **"Set A is a subset of Set B"**. If set B is a subset of A only, output **"Set B is a subset of Set A"**. If none of the above are true, output **"Neither set is a subset of the other. Their intersection is:"** followed by the intersection of the two sets in numerical order. Intersection output is to be comma separated and enclosed in curly braces. If the intersection has no elements the intersection is the empty set {}. All output for each test case should be printed on one line per each case.

Sample Input:

```
7
A={1,2,3,4,5,6,7,8}
B={3,2}
A={1,2,3,6,7,9,11,12}
B={3,2,1}
A={2,3}
B={2,1,3}
A={1,3}
B={2,1}
A={1,2,3,4,5}
B={9,8,7,6}
A={1,3,5,7,9}
B={3,1,9,5,7}
A={1,4,5}
B={5,1,6}
```

Sample Output:

```
Set B is a subset of Set A
Set B is a subset of Set A
Set A is a subset of Set B
Neither set is a subset of the other. Their intersection is: {1}
Neither set is a subset of the other. Their intersection is: {}
Set A is equal to Set B
Neither set is a subset of the other. Their intersection is: {1,5}
```

8. Olivia

Program Name: Olivia.java **Input File: olivia.dat**

Olivia is getting bored memorizing multiplication tables. The tables Olivia must memorize have N rows and M columns. The top-most row is row 1, and the left-most column is column 1. At the intersection of the R-th row and the C-th column is the value R * C.

To make working with the table more fun, Olivia likes starting at some cell in the table, and only moving down and right, exploring the numbers as she goes. Olivia's favorite number is K, and she enjoys it if she can get to a cell with value K from the starting cell.

After watching Olivia play for a little while, her parents were wondering how many different cells there are on an N by M multiplication table where you can walk to some cell with value K by only moving right and down. Since these tables can get massive, they've asked you to write a program to help them out.

Input: The first line of input is an integer T ($1 \leq T \leq 50$), the number of test cases. Each test case contains three space-separated integers N, M, and K. ($1 \leq N, M, K \leq 1{,}000{,}000{,}000$). N is the number of rows in the multiplication table, M is the number of columns in the multiplication table, and K is the target number.

Output: For each test case, output a single integer: the number of cells where there is at least one path from that cell to a cell with value K.

Sample input:
```
4
4 3 2
4 3 6
10 10 25
3141592 6535897 9323846
```

Sample output:
```
Case #1: 3
Case #2: 8
Case #3: 25
Case #4: 71686489
```

Sample explanation:
For the input `4 3 2`, the table is:

1	2	3
2	4	6
3	6	9
4	8	12

The "fun" cells are the green 1 and 2 in the top row, and the green 2 in the second row. For the input `4 3 6`, the table is the same as above, but any green or blue cell is "fun". These are the first two rows and the first two cells in the third row.

In the third case, any cell in the first five rows and the first five columns is a valid starting point.

9. Randy

Program Name: Randy.java **Input File: randy.dat**

Randy loves to go bowling, and needs your help writing a program to calculate the final score of a game, given a full score card. The final score is the sum of all 10 frames according to the following standard rules of bowling.

Strike (Denoted as X)
If you knock down all 10 pins in the first shot of a frame, you get a strike. No further shot is rolled for the current frame.
How to score: A strike earns 10 points for the frame, plus the sum of the pins knocked down in the next *two* shots in following frames.

Spare (Denoted as /)
If you knock down all 10 pins using both shots of a frame, you get a spare.
How to score: A spare earns 10 points for the frame, plus the number of pins knocked down in the first shot of the next frame.

Open Frame
If you do not knock down all 10 pins using both shots of your frame (9 or fewer pins knocked down), you have an open frame.
How to score: An open frame only earns the number of pins knocked down in that frame.

The 10th Frame
The 10th frame is a bit different:
If you roll a strike in the first shot of the 10th frame, you get 2 more shots immediately.
If you roll a spare in the first two shots of the 10th frame, you get 1 more shot immediately.
If you leave the 10th frame open after two shots, the game is over and you do not get an additional shot.
How to Score: The score for the 10th frame is the total number of pins knocked down for all the rolls in the 10th frame.

Input: Input starts with a line containing an integer N (1<=N<=10), the number of test cases. The following N lines consist of a single player's full score card from a complete game of bowling. An X represents a strike, and a / represents a spare. All inputs will be valid, complete games. No validation of input is needed.

Output: For each test case, output the final score of that game.

Sample Input:

```
10
X9/406/X90XX636/X
XXXXXXXXXXXX
53628180908160819052
X639/XXX5/8/6/7/X
0/0/0/0/0/0/0/0/0/0
X0/0/X0/00XXX0/X
XXXXXXXXXXX9
9/06X8/3/X06367/X62
81X9/XX903/XXX5/
X4/30XX42X9/7/XXX
```

Sample Output:

```
160
300
82
194
100
170
299
138
201
169
```

10. Sergey

Program Name: Sergey.java **Input File: sergey.dat**

Sergey has taken a job with ASCII Art Unlimited Inc. AAUI trades in ASCII art images. Often times a customer will ask that a piece of artwork that they would like to purchase be shown sideways or upside down. AAUI has some wonderful artist creating beautiful images but they haven't yet developed the technology to rotate images. That's why they hired you! Write a program that will rotate and display an ASCII art image a designated number of degrees.

Input: The first line in the input file will be a single number N which will not be greater than 10. N is the number of ASCII images that should be rotated. N sets of data will follow. Each data set will begin with a line that contains two numbers, L and D. L is the dimension of one side of the image and 2 <= L <= 25. Each image will always be square. D is the number of degrees that the image should be rotated in a clockwise fashion and 90 <= D <= 720. D will always be a multiple of 90. The images will consist of only printable, non-space characters.

Output: Each rotated image followed on the next line by the word DONE in all caps. There should not be any space between DONE and the next rotated image.

Sample input:

```
2
5 90
<...>
.O.O.
..|..
.(_).
.....
22 180
**********************
**********************
**********************
**********************
**********************
* _********************
(.\********************
*).)**__--__**********
(.(**/......\**A_-_A**
*\.\/........\/.,.,.\*
**\..........=;..t../=
***\....____...\_--/**
****\..|****\..;******
****/.//****|.||******
***/_,))****|_,))*****
**********************
**********************
**********************
**********************
**********************
**********************
**********************
```

Sergey sample output:

```
....<
.(.O.
._|..
.).O.
....>
DONE
**********************
**********************
**********************
**********************
**********************
**********************
**********************
*****)),_|****)),_/***
******||.|****//./****
******;..\****|..\****
**/--_\...____....\***
=/..t..;=..........\**
*\.,.,./\........./\.\*
**A_-_A**\....../**(.(
**********__--__**).)*
********************\.(
********************_*
**********************
**********************
**********************
**********************
**********************
DONE
```

11. Terry

Program Name: Terry.java **Input File: terry.dat**

Inspired by White Collar, Ocean's 12, and the Rick and Morty Heist episode, Terry is looking to change careers from computer science student to international art thief. Career changes are usually hard, but because Terry joined a crew of 10 skilled thieves, the transition to a life of crime isn't as intense as it otherwise would have been.

For Terry's first heist, the syndicate has targeted the Ultimate Imaging Laboratory (UIL). The UIL is currently restoring N different paintings. At night, no one is there except an inattentive security guard, and each painting is guarded by a unique security system.

With over 200 years of combined thievery experience, your syndicate can accurately determine how much each work of art can sell for on the black market, and the probability the security system detects your intrusion, given as a percentage. These values for the paintings are denoted as V_i and P_i, respectively. For this heist to be worth your while, you need to steal at least K dollars' worth of art.

Terry decided a good way to use his skills from his past life would be to write a program to figure out which paintings to steal to minimize the probability of getting caught. He will get caught if even one alarm goes off. All alarm systems operate independently from each other. Remember, the probability of two independent events occurring is the product of the probabilities of each event happening. Can you help Terry write the program?

DISCLAIMER: The University Interscholastic League does not endorse using computing skills for breaking the law or other evil misdeeds.

Input: Input starts with a line containing an integer T ($1 \leq T \leq 20$), the number of test cases. The first line of each test case contains two integers N and K. Then follow N lines, each with integers V_i and P_i.

$1 \leq N \leq 100$
$1 \leq K, V_i \leq 10{,}000$
$0 \leq P_i \leq 100$

Output: For each test case, output the case number and the minimum possible percentage of getting caught while stealing at least K dollars' worth of paintings. Output the percentage rounded to exactly six decimal places. Format the output as in the samples.

Sample input

```
3
3 2
5 50
3 10
8 25
2 10
6 50
7 50
3 141
59 65
89 79
32 38
```

Sample output

```
Case #1: 10.000000
Case #2: 75.000000
Case #3: 92.650000
```

Sample Explanations:

In the first sample, if you steal any single painting, that's sufficient. To minimize the odds of getting caught, steal the painting of value 3, as the odds of getting caught are only 10%.

In the second sample, regardless of the order he decides to steal the paintings, Terry needs to steal both paintings. There is a 50% chance of getting caught on the first painting and 50% * 50% = 25% chance of getting caught on the second painting. Therefore, the overall odds of getting caught are 75%.

12. Yana

Program Name: Yana.java **Input File: yana.dat**

Yana has devised a new encryption scheme and asked your UIL programming team to create a program to test her idea. It starts with an 8 x 8 grid that contains the digits 0…9, the uppercase letters A…Z, and the lowercase letters a…z, the ASCII space (shown as a -), and the period. They are initially arranged as shown below in the left grid.

Her technique uses 9 non-negative integers, 1 for each column and rotates each column down that number of positions with the characters at the bottom rotating up to the top positions like spinning wheels. The right grid shows the result of rotations for: 1, 11, 21, 7, 18, 4, 14, 8.

0	8	G	O	W	d	l	t
1	9	H	P	X	e	m	u
2	A	I	Q	Y	f	n	v
3	B	J	R	Z	g	o	w
4	C	K	S	-	h	p	x
5	D	L	T	a	i	q	y
6	E	M	U	b	j	r	z
7	F	N	V	c	k	s	.

7	D	J	P	b	h	n	t
0	E	K	Q	c	i	o	u
1	F	L	R	W	j	p	v
2	8	M	S	X	k	q	w
3	9	N	T	Y	d	r	x
4	A	G	U	Z	e	s	y
5	B	H	V	-	f	l	z
6	C	I	O	a	g	m	.

Encoding of messages are done my substituting the characters in the right grid for characters from the left grid. Examples:

- "UIL Region 2020" becomes "VLGYSikeqpY1717"
- "The Art of Programming Vol. 1 1968" becomes "UdiYFltYqjYQlqklZooepkYOqn.Y0Y0E5D"

Input: First line contains N ≤ 10, the number of test cases. Each test case starts with a line of 9 comma-separated integers. The first 8 are non-negative integer rotation factors with values < 100, in order of columns 1…8. The last one is the number of messages ≤ 10 that follow, each on a separate line and no longer than 100 characters in length. The messages will be limited to the characters shown in the grids and will not have trailing spaces.

Output: The test case number and the message number of that test case separated by a colon and inside a pair of brackets. That is then followed by two dashes and the encoded message in a pair of quotation marks. Following each test case, display a line containing exactly 10 equal signs "==========".

Sample input:

```
2
1,11,21,7,18,4,14,8,2
UIL Region 2020
The Art of Programming Vol. 1 1968
22,15,42,8,13,36,17,27,3
4763Z NORTH SQ.
Bleep squad
Junk CODE
```

Sample output:

```
[1:1]--"VLGYSikeqpY1717"
[1:2]--"UdiYFltYqjYQlqklZooepkYOqn.Y0Y0E5D"
==========
[2:1]--"6105bcLORTNcSQx"
[2:2]--"CsiiocrpzWh"
[2:3]--"HzmgcDOEF"
==========
```