2015 WVU Tech High School Programming Competition

Department of Computer Science and Information Systems
West Virginia University Institute of Technology

November 14, 2015

Instructions

1. There are ten (10) problems in the packet, using letters A-J. These problems are NOT sorted by difficulty.

2. All solutions must read from standard/console input and write to standard/console output.

3. Solutions for problems submitted for judging are called runs. Each run will be judged. Runs for each particular problem will be judged in the order in which they are received. However, it is possible that runs for different problems may be judged out of order. For example, you may submit a run for B followed by a run for C, but receive the response for C first. DO NOT request clarifications on when a response will be returned. If you have not received a response for a run within 10 minutes of submitting it, **you may ask the lab monitor to send a runner to the judge to determine the cause of the delay. Under no circumstances should you ever submit a clarification request about a submission for which you have not received a judgment.**

   The judges will respond to your submission with one of the following responses. In the event that more than one response is applicable, the judges may respond with any of the applicable responses.

Response	Explanation
YES	Your submission has been judged correct.
NO - Compile Error	Your submission didn't compile correctly.
NO - Time Limit Exceeded	Your submission ran for longer than 60 seconds without terminating.
NO - Wrong Answer	Your submission produced an incorrect answer for at least one of the test cases.
NO - Runtime Error	Your submission terminated abnormally.
NO - Output Format Error	The output of your program does not match the correct output.
NO - Other (Please see staff)	Your submission produced an error that is not classified from above. The judge will provide additional information if this happens.

4. A team's score is based on the number of problems they solve and penalty points. The penalty points reflect the time required to solve a problem correctly and the number of incorrect submissions made before the problem is solved. For each problem solved correctly, penalty points are charged equal to the time at which the problem was solved plus 20 minutes for each incorrect submission. No penalty

points are added for problems that are never solved. Teams are ranked by the number of problems solved. Ties are resolved in favor of the team with the fewest penalty points.

5. Each problem contains sample input and output. However, you may be assured that the judges will test your submission against several other more complex data sets, which will not be revealed. Before submitting your run, you should design other input sets to fully test your program. Should you receive an incorrect judgment, you are advised to consider what other data sets you could design to further evaluate your program.

6. In the event that you think a problem is ambiguous, you may request a clarification. Read the problem carefully before submitting a clarification. If the judges believe that the problem statement is sufficiently clear, you will receive the response, "The problem statement is sufficient. No clarification is necessary." If you receive this response, you should read the problem description more carefully. If you still think there is an ambiguity, you will have to be more specific or descriptive of the ambiguity you have found. If the problem statement is ambiguous in specifying the correct output for a particular input, please include that input data in the clarification request.

7. The submission of abusive programs or clarification requests to the judges will be considered grounds for immediate disqualification.

8. Good luck and have fun!

Problem A

Calculating Change

As a cashier, a potential issue is giving incorrect change to a shopper. Instead of seeing the amount of change to return, it would be easier to know the amount and type of bills and coins to return. Your task in this problem: Given an amount, determine the fewest number of bills and coins needed to add to the given amount.

Input

Each test case is described by one line of input having the format: `number`. Each dollar amount ranges from $0.01 to $1,000,000.00. If a line of input consists of the word `done`, the program should exit.

Output

For each case, display the case number followed by the fewest number of each of the necessary types of bills (100, 50, 20, 10, 5, 1) and fewest number of each of the necessary types of coins (.50, .25, .10, .05, .01) to add to the given amount. Each dollar/coin amount must be printed on a separate line. Any bills/coins not used must be omitted from the output.

Sample Input	Sample Output
156.35 345.76 done	Case 1: 1 100 Bill(s) 1 50 Bill(s) 1 5 Bill(s) 1 1 Bill(s) 1 25 Cent Piece(s) 1 10 Cent Piece(s) Case 2: 3 100 Bill(s) 2 20 Bill(s) 1 5 Bill(s) 1 50 Cent Piece(s) 1 25 Cent Piece(s) 1 1 Cent Piece(s)

Problem B

Adding Big Integers

No matter how good our computer is, sometimes we need to do arithmetic with very large numbers, and our computer cannot store these numbers. In this problem, you will be adding two integer numbers which are larger than the computer can store.

Input

Each test case is described by one line of input consisting of two integer values. Each number can have up to 50 digits and is separated by a space. A test case of `0 0` indicates the end of input.

Output

For each case, display the case number followed by the sum of the two integer values.

Sample Input

```
34567898765432185357 98712345678976543245
765432123456789876543212456789876343454335534 12309876543234567898765432124670 0
```

Sample Output

```
Case 1:  133280244444408728602
Case 2:  765432123456791107530866780246666219997548001
```

Problem C

Catalan Numbers and Parentheses

Matrix multiplication[1] is associative, but not commutative. Therefore, for any matrices A, B, C, and D,

$$A(B(CD)) = A((BC)D) = (AB)(CD) = (A(BC))D = ((AB)C)D$$

Those are all of the possible different ways to compute the product $ABCD$.

In this problem, you are asked to find $P(n)$, which is the number of different ways a product of n matrices can be parenthesized. We will let $P(1) = 1$, and it should be clear that $P(2) = 1$, $P(3) = 2$, $P(4) = 5$.

Hint: Suppose that the parentheses associated with the last multiplication is given by

$$(A_1 A_2 \dots A_k)(A_{k+1} A_{k+2} \dots A_n).$$

The number of different ways to parenthesize $A_1 A_2 \dots A_k$ is $P(k)$, and the number of different ways to parenthesize $A_{k+1} A_{k+2} \dots A_n$ is $P(n-k)$. Therefore, if the parentheses for the last multiplication is as shown, there are $P(k)P(n-k)$ different ways to completely parenthesize the product.

Input

Each line of input consists of a single integer n, where $1 \leq n \leq 36$. An input of `0` indicates the end of the input.

Output

For each line of input, the output will be a single integer representing $P(n)$.

Sample Input	Sample Output
1 2 3 4 6 10 25 0	1 1 2 5 42 4862 1289904147324

[1] You do not have to know anything about matrices or multiplication of matrices to solve this problem.

Problem D

Money, Money Money

Suppose you are given n cents. Find the number of ways n cents can be paid using the following:

Coins:

- Pennies
- Nickels
- Dimes
- Quarters

Paper Currency:

- $1 Bills
- $5 Bills
- $10 Bills
- $20 Bills
- $50 Bills
- $100 Bills

Input

Each line of input consists of a single integer n, where $0 < n \leq 10000$. An input of 0 indicates the end of the input.

Output

For each line of input, the output will be a single integer representing the number of ways n can be represented using the given currencies.

Sample Input	Sample Output
1 2 10 25 50 100 5000 0	1 2 4 13 49 243 888601247

Problem E

Column Transposition

A Column Transposition is a simple cipher algorithm. Begin by selecting an encoding word (of any length). Create a grid with the number of columns being equal to the length of the word selected. Write the word across the top row of the grid.

Below the word, fill in the message you wish to encode. You should go across each row and "wrap" down to the next row when you reach the end. (Spaces included in the message are included in the grid). Sort the columns using the letters in the top row (i.e., the encoding word). The encoded message is formed by listing all the letters in the first row and then the next row, and so forth.

The object of this problem is to write a program that will encode any message by a word given by the user.

For example:

Word: `hatter`

Message: `We are all mad here. But those are the best kinds of people.`

Original:

h	**a**	**t**	**t**	**e**	**r**
W	e		a	r	e
	a	l	l		m
a	d		h	e	r
e	.		B	u	t
	t	h	o	s	e
	a	r	e		t
h	e		b	e	s
t		k	i	n	d
s		o	f		p
e	o	p	l	e	.

Sorted:

a	**e**	**h**	**r**	**t**	**t**
e	r	W	e		a
a			m	l	l
d	e	a	r		h
.	u	e	t		B
t	s		e	h	o
a			t	r	e
e	e	h	s		b
	n	t	d	k	i
		s	p	o	f
o	e	e	.	p	l

Encoded Message: `erWe aa mlldear h.uet Bts ehoa treeehsb ntdki spofoee.pl`

Input

Each line entered via the console will contains a word, a semicolon (`';'`), and a message that needs to be encoded. The message can contain uppercase letters, lowercase letters, blank spaces, and symbols. If the line contains a `99`, the program should exit.

Output

The output should display the the encrypted message on a single line.

Sample Input

```
hatter;We are all mad here. But those are the best kinds of people.
lion;ZEBRAS ARE COOL!
99
```

Sample Output

```
erWe aa  mlldear h.uet Bts ehoa  treeehs b ntdki  spofoee.pl
EZRBSAA ERC OO!L
```

Problem F

Times Change

This program will ask the user to enter a time (either 12 hour or 24 hour format). The program will need to read the user console input and convert it to the alternative format. If the user enters a value greater than the standard format, the output will need to have the "correct" time. For example, if the user enters 72 seconds (a value greater than the 60 second format), you will need to fix it by "incrementing" the minutes by 1 and the seconds would be set to 12. Note: you may need to increment other values and change between AM and PM.

Input

Each line will represent two values separated by a colon (`':'`). The first value (12 or 24) will indicate the format of the time entered. The second value is the time and will be given in `hours:minutes:seconds`. (Note: each part of the input is separated by a colon, `':'`) If the format is 12 hours, the input line will also include an `A` or `P` at the end. If the line contains a `99`, the program should exit.

Output

For each input, the output should display the time in the alternative format. For example, if the input is in a 12-hour format, the output should be in the 24-hour format. The time format should be in `hours:minutes:seconds`. If the output format is 12 hours, an `AM` or `PM` should be included, separated by a space.

Sample Input	Sample Output
12:4:7:9:A 24:14:23:57 12:11:59:104:P 99	4:7:9 2:23:57 PM 0:0:44

Problem G

NoRays

An array is one of the most basic data structures in programming. One use for an array is to display a set of values. For example, we could display an array of characters as `['a', 'a', 'c']`.

In this problem, you will look at NoRays. These are like arrays except they don't contain any duplicates. For example, the NoRay of our earlier example would be `['a', 'c']`. Note that the term "NoRay" is not a real programming term. We're simply using it for the purpose of this problem.

Your task is to convert an array of characters with duplicates into NoRays.

Input

The first line will contain an integer N, where $1 \leq N \leq 20$, that represents the number of test cases to follow. Each test case will be on its own line and will be at least 4 characters (with a maximum of 20). Each element of the array will be followed by a space.

Output

The output will consists of a single line which is the NoRay of the given input. In other words, the output should be the contents of the original array without any duplicates. Each element of the NoRay will be separated by a space. Each letter should be in the order they are presented.

Sample Input	Sample Output
4 a b b a b a c d e f b a b b a g a b b b a g	a b b a c d e f b a g a b g

Problem H

Set Intersection

Let A and B be two sets of integers. The intersection of sets A and B, denoted as $A \cap B$, is the set of integers that are in both A and B. For example, if $A = \{1, 3, 5, 7, 9, 11\}$, and $B = \{2, 3, 5, 7, 11\}$, then $A \cap B = \{3, 5, 7, 11\}$. You are asked to write a program that computes the intersection of two sets.

Note: If you use the naive approach for solving this problem, your program will encounter the **NO - Time Limit Exceeded** error.

Input

Each test case consists of three lines of input. The first line contains two integers N and M, where $1 \leq N, M \leq 100000$. N is the number of items in set A, and M is the number of items in set B. The second line contains N integers, in increasing order, containing all of the items in set A. The third line contains M integers, in increasing order, containing all of the items in set B. Note that each item in both A and B can range from 1 to 500000. You may assume that each integer in set A is unique, and each integer in set B is unique. If $N = 0$ and $M = 0$, then the program should terminate.

Output

For each test case, the output should be a single line containing all of the integers in both A and B. The integers should be displayed in sorted order and separated by a single space. If there are no integers in both A and B, then the output should be `0`.

Sample Input	Sample Output
6 5 1 3 5 7 9 11 2 3 5 7 11 2 3 1 2 3 4 5 0 0	3 5 7 11 0

Problem I

Bullseye

Suppose you are playing a game of darts. The object to the game is to throw a dart on a circular board. This board is divided into rings. The closer your dart gets to the middle ring, you'll get a higher number of points. Hitting the middle ring (i.e., getting a Bullseye) gives you the maximum number of points. You want to know how many points to expect each time you throw a dart.

To compute the expected number of points, you will need two formulas. The first formula computes the expected value, which is

$$E = x_1p_1 + x_2p_2 + x_3p_3 + \cdots + x_ip_i + \cdots + x_np_n$$

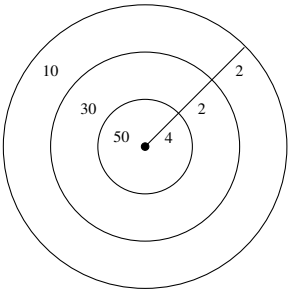
where n is the number of possible values, each x_i represents a possible value, each p_i represents the corresponding probability, and $1 \leq i \leq n$. The second formula is computing the area of a circle, which is

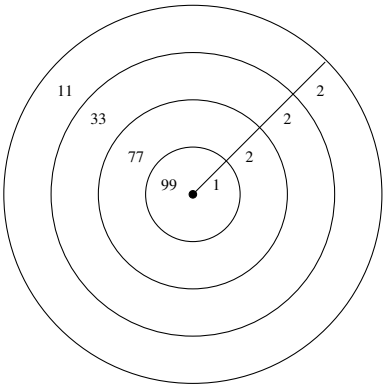
$$A = \pi r^2$$

where r is the radius of a circle.

Finally, note that the probability of the dart landing in a certain region is the area of that region divided by the area of the smallest region containing all possible landing points. To simplify the problem, we are concerned only with dartboard that have concentric circles, where higher values are closer to the center, and we will ignore the possibility of missing the dartboard entirely.

Examples of dartboards are given below:





Input

The first line of input is a single integer N, where N is the number of test cases. For each test case, the first line of input is an integer M, representing the number of successive rings on the dartboard, including the Bullseye. Each of the remaining M lines contains two integers x and y. The first integer x is the radius of a ring ($0 < x$), and the second integer y is the point value of the ring ($0 < y$). Note that the point value is equal to the point value associated with the smallest circle in which the dart lands (largest possible value).

Output

For each test case, the output should be a single line containing the expected value.

Sample Input	Sample Output
2 3 4 50 6 30 8 10 4 1 99 3 77 5 33 7 11	26.250 30.755

Problem J

Let's Go Bowling

A game of bowling consists of 10 frames. In each of the first nine frames, you have 2 rolls to knock down 10 pins. If you knock all 10 pins down on the first roll, you get a "strike," and you are not given a second roll for that frame. If you knock all 10 pins down with two rolls, you get a "spare."

The score for each frame depends on whether or not you get a strike or a spare. If you get a strike, you get 10 points plus the number of pins knocked down on the next two rolls (in the next frame or two frames if you get a strike on the next roll). If you get a spare, you get 10 points plus the number of pins knocked down on the next roll (in the next frame).

On the 10th frame, the rules are slightly different. If you get a strike, you get 2 more rolls, totaling 3 rolls for the frame. If you get a strike on the second roll, your third roll will consist of 10 new pins. If you do not get a strike on your second roll, your third roll consists of the remaining pins from the second roll. The score for your third roll will be the number of pins knocked down.

If you get a spare on the 10th frame, you get 1 more roll consisting of 10 pins. The score for your third roll will be the number of pins knocked down.

The table below gives an example of a game and the final score:

Frame	Notation	Points	Total
1	`8 1`	$8+1=9$	9
2	`X`	$10+10+3=23$	32
3	`X`	$10+3+7=20$	52
4	`3 /`	$10+4=14$	66
5	`4 5`	$4+5=9$	75
6	`1 /`	$10+10=20$	95
7	`X`	$10+10+3=23$	118
8	`X`	$10+3+7=20$	138
9	`3 /`	$10+4=14$	152
10	`4 / 8`	$4+6+8=18$	170

Your job is to compute the score for a game of bowling given the scores for each of 10 frames. A strike is represented by an `"X"`, a spare is represented by the number of pins knocked down in the first roll followed by a slash (`"/"`), and an open frame is represented by 2 integers: The number of pins knocked down in each roll. Similarly, the 10th frame is represented either as:

1. `X X X`
2. `X X a`
3. `X a /`
4. `X a b`
5. `a / X`

6. `a / b`

7. `a b`

where `a` and `b` are integers between 0 and 9.

Input

Each test case consists of at least two lines of input. The first line contains an integer N indicating the number of games. Each of the remaining N lines contains the scores for a game. Each score (separated by a space) is either a digit between 0 and 9, an `X`, or a `/`. Note that the frames are not delimited.

Output

For each game, a single number is produced as output. This number is the final score for the corresponding game. Each score should be on a separate line.

Sample Input	Sample Output
3 2 6 5 0 4 4 0 9 3 4 8 1 1 4 3 2 5 4 6 1 3 / 4 4 X 5 3 9 0 1 / X 8 / 7 2 X 4 3 8 1 X X 3 / 4 5 1 / X X 3 / 4 / 8	72 140 170