SEVENTH ANNUAL JUILFS CONTEST
SPRING 2015

Dr. Seuss

Department of Computer and Information Science
University of Oregon

2015 May 30

contributors: Skyler Berg, Atlee Brink, Chris Wilson

A: RINGS (13 POINTS)

How much mass is contained in a system of rings around a planet? A team of astronomers has collected data about the size and average density of rings around various planets. Now they are arguing about whether there is enough mass in all the rings of a single planet to make a decent moon. You know, assuming the rings could be scooped up and formed into a moon-shape.

Nobody wants to see astronomers fight, so we should help them resolve their argument peacefully. We have access to the data they collected, but we need to perform some arithmetic to see how much total mass is around each planet whose rings they surveyed.

The data is organized by planet, and includes its number of rings, the inner and outer radius of each ring, and the average density of each ring. For simplicity, we can assume that the mass of one ring is its two-dimensional area times its average density. Thus we need to calculate the mass of each ring, and then output the total mass of all a planet's rings. They have collected data about several planets, so we will need to do this several times.

On a technical note, the astronomers have insisted that we use exactly 3.14159 as the value of Pi in our calculations. Apparently there is an argument about the significance of the other digits, and we want to remain neutral.

Input:
The first line of input is the number of planets 1 <= P <= 50. Then, for each planet, the first line is the number of rings 1 <= R <= 20. Then, for each ring, there is one line containing three real numbers separated by spaces: the inner radius of the ring, the outer radius of the ring, and the average density of the ring.

Output:
For each planet, output the total mass in all of its rings. The astronomers are tired of looking at dots, so we should truncate (floor) each number to an integer value. For example, 123.8 should be output as just 123 , with no decimal point. Each number should be on its own line.

Sample Input:

```
3
1
3 5 1
3
5.2 17.1 3.1
18.9 19.1 0.9
25.3 26.8 0.02
2
100.1 100.2 1.62
100.5 100.6 2.11
```

Sample Output:

```
50
2610
235
```

B. Count the Cats in the Hats (12 points)

The Cat in the Hat has made a terrible mess of things! He was trying to help the Sneetches clean out a peculiar machine when he raised his hat, which had under it Little Cat A. He asked Little Cat A to help, and after two minutes, the Little Cat raised his hat, revealing another Little Cat, with more Little Cats in its hat!

Each Little Cat releases another Little Cat after two minutes and then releases another Little Cat every minute thereafter. Little Cat A is always considered to have been released at in minute 1. So in minute 1, there is 1 Little Cat. In minute 2, there is still 1 Little Cat. In minute 3, Little Cat A releases Little Cat B and there are 2 Little Cats. In minute 4, Little Cat A releases Little Cat C and there are 3 Little Cats. In minute 5, Little Cat A releases Little Cat D, Little Cat B releases Little Cat E, and there are 5 Little Cats. And so on and so forth.

During certain minutes the Cat in the Hat manages to catch a Little Cat and stuff it back into his hat. If the Cat in the Hat catches a Little Cat during minute 5, then there would be 4 rather than 5 Little Cats out and about at that minute. You may assume that the Cat in the Hat always catches a newly released Little Cat. Thus in the next minute there would be 7 Little Cats.

== input ==
The first line of input will contain a single integer, T, followed by T test cases. Each test case will consist of a normal start with a line contain two space separated integers, 1 <= M <= 30 and 1 <= N <= M. M represents the number of minutes that pass and N represents the number of Little Cats that the Cat in the Hat catches and puts back in his hat. The next line will contain N space separated integers representing each minute that Cat in the Hat catches a Little Cat.

== output ==
For each test case print the number of Little Cats out and about after N minutes have passed. The output will be greater than or equal to 0 and less than 2^31.

Sample Input	Sample Output

3
6 1
5
10 3
3 6 9
1 1
1
	7
	28
	0

C: DO THE RANKINGS AGREE? (12 POINTS)

Your team has been retained by the director of a competition who supervises a panel of judges. The competition asks the judges to assign integer scores to competitors – the higher the score, the better. Although the event has standards for what score values mean, each judge is likely to interpret those standards differently. A score of 100, say, may mean different things to different judges.

The director's main objective is to determine which competitors should receive prizes for the top positions. Although absolute scores may differ from judge to judge, the director realizes that relative rankings provide the needed information – if two judges rank the same competitors first, second, third, ... then they agree on who should receive the prizes.

Your team is to write a program to assist the director by comparing the scores of pairs of judges. The program is to read two lists of integer scores in competitor order and determine the highest ranking place (first place being highest) at which the judges disagree.

The first line of input will be a number C, the number of competitor pairs. What follows will be a series of C score list pairs. Each pair begins with a single integer giving the number of competitors *N*, 1 < *N* < 1,000. The next *N* integers are the scores from the first judge in competitor order. These are followed by the second judge's scores – *N* more integers, also in competitor order. Scores are in the range 0 to 100,000 inclusive. Judges are not allowed to give ties, so each judge's scores will be unique. Values are separated from each other by one or more spaces.

For each score pair, print a line with the integer representing the highest-ranking place at which the judges do not agree. If the judges agree on every place, print a line containing only the word 'agree'. Use the format below: "`Case`", one space, the case number, a colon and one space, and the answer for that case with no trailing spaces.

Sample Input

```
2
4
3 8 6 2
15 37 17 3
8
80 60 40 20 10 30 50 70
160 100 120 80 20 60 90 135
```

Sample Output

```
Case 1: agree
Case 2: 3
```

D: FISHY MATH (10 POINTS)

The cat sees fish. The cat must do math on the fish since the cat is not allowed to eat them. But in Fish World the numbers come before the thing to do with the number, such as “3 5 *”. This says apply the * to the 3 and the 5, which evaluates to 15. This is called *postfish* notation.

A *postfish* expression will involve numbers and the operators + (plus), * (times), and – (minus) separated by spaces. Any operator is to be applied to the previous two values that have been seen or computed. For example, “3 5 *” is 15, “3 5 * 2 +” is 17, and “3 5 2 * +” means 3+(5*2) = 13.

The first line of input will be an integer C, the number of expressions. This will be followed by C *postfish* expressions. For each expression, your program will need to print the single integer it evaluates to.

Each expression will properly evaluate to a number: there are no errors like missing operators or missing or out of order numbers.

Sample input:

```
3
5 4 + 3 -
100 2 - 3 +
1 2 3 4 + * +
```

Sample output:

```
6
101
15
```

E: SIMPLE STRING COMPRESSION (6 POINTS)

When you live in a shell, you have to keep all your strings of characters short. To do this, you have devised a very basic string compression method. The idea is that when there is a series of a character being repeated several times, we will write that character just once, followed by the number of repetitions.

For example, aaaabb would become a4b2, which is a little bit shorter. aaaaaaaaaaaa becomes a12, which is a lot shorter. On the other hand, abcd becomes a1b1c1d1, which is longer. Our method is not perfect, but we need a program for it anyway.

The first line of input will be an integer C, the number of strings to compress. This will be followed by C strings, each on a separate line. The strings consist of lower case characters [a-z], with no punctuation or spaces. All have length at least 1.

Your output should consist of the compressions of each of the input strings, each on its own line.

Sample input:

```
3
aaaaaabbbbbc
aabcdeeeeeeeee
abcdefghij
```

Sample output:

```
a6b5c1
a2b1c1d1e9
a1b1c1d1e1f1g1h1i1j1
```