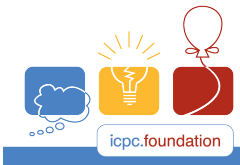
2017 ECNA Regional Practice Contest

ECNA 2017

Practice Session

October 27



2017 ACM ICPC East Central North America Regional Contest

Problems

A Arggggggh!
B Every Second Counts
C Is Everybody Appy?
D KaBlam!
E Life Savings
F Treating Fractions Irrationally

Do not open before the contest has started.

ACM International Collegiate Programming Contest
2017 East Central Regional PRACTICE Contest
Grand Valley State University
University of Cincinnati
University of Windsor
Youngstown State University
October 27, 2017

Sponsored by IBM

Rules:

1. There are **six** problems to be completed in **90 minutes**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. When displaying results, follow the format in the Sample Output for each problem. Unless otherwise stated, all whitespace in the Sample Output consists of exactly one blank character.
4. The allowed programming languages are C, C++, Java, Python 2 and Python 3.
5. All programs will be re-compiled prior to testing with the judges' data.
6. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
7. Programs will be run against multiple input files, each file containing a single test case.
8. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
9. All communication with the judges will be handled by the Kattis environment.
10. Judges' decisions are to be considered final. No cheating will be tolerated.

Problem A
Arggggggh!

Congratulations! You've just inherited a treasure map from a long deceased ancestor who just happened to be a pirate. With visions of gold doubloons and jeweled cutlasses dancing in your head you carefully unfold the map. However instead of the standard picture map with a large X where the treasure is, this map is simply a list of instructions on where to start digging. The first instruction is the starting location, given as two integers x y giving the starting point's latitude and longitude. After this are instructions like N 7 or SW 6 which specify to walk 7 feet north or 6 feet southwest. After following all of these instructions you will be located over the spot to start digging.

(Just to be clear, moving north increases the y value of your location and keeps the x value constant; moving east increases the x value of your location and keeps the y value constant. You can determine the effects the other directions have on the location.)

Now, you could go through the laborious task of executing each of the instructions one at a time, starting at the initial location, but it would be a lot easier to just calculate the final location and head right there. And that's the task for this problem.

Oh, one more thing: turns out your ancestor was actually a Pittsburgh Pirate, and the treasure consists of a bunch of old baseballs and a really moldy mitt. I'm sure you can get a doubloon or two for them on eBay.

Input

Input starts with a line containing a single integer n ($1 \leq n \leq 100$) indicating the number of instructions. The next n lines each contain a single instruction. The first instruction consists of two integers x y ($0 \leq x, y \leq 100$) indicating the starting point. The remaining instructions are of the form `dir` d, were `dir` is the direction (either `N`, `NE`, `E`, `SE`, `S`, `SW`, `W` or `NW`) and d ($1 \leq d \leq 100$) is the distance to walk.

Output

Display the x and y coordinates of the final location, separated by a single space. All coordinates should have a maximum relative or absolute error of 10^{-6}.

Sample Input 1	Sample Output 1
3 5 5 N 10 W 7	-2.00000000 15.00000000

Sample Input 2	Sample Output 2
3 0 0 NE 10 SE 7	12.02081528 2.12132034

Problem B
Every Second Counts

Meredith runs a taxi service called Ruber which offers rides to clients in small towns in western Pennsylvania. She wants to get every possible dime out of people who use her taxis, so her drivers charge a flat fee not per minute but per second. It's important, therefore, to be able to determine the exact amount of elapsed time between the moment a client enters a cab until the moment they leave. Trying to write a program to do this has driven Meredith crazy (pun intended) so she's come to you for some help.

Input

Input consists of two lines: the first contains the start time and the second contains the end time for a single taxi ride. Each time is of the form $hh : mm : ss$, giving the hour, minute and seconds. Meredith uses a 24 hour clock, with $0 : 0 : 0$ representing 12 midnight and $23 : 59 : 59$ representing one second before midnight. Note that the end time may have a value less than the start time value if the ride spans midnight (see the last sample test case for an example of this).

Output

Display the number of seconds between the two times. No cab ride will be equal to or longer than 24 hours.

Sample Input 1	Sample Output 1
10 : 0 : 0 11 : 0 : 0	3600

Sample Input 2	Sample Output 2
13 : 30 : 52 13 : 31 : 7	15

Sample Input 3	Sample Output 3
23 : 0 : 0 1 : 30 : 0	9000

This page is intentionally left blank.

Problem C
Is Everybody Appy?

Joyce Stick is a mother of several children and has strict rules on how much technology they can use. One of her rules is that each kid can have at most one app on their phone. While the kids aren't thrilled with this rule, they know if they complain they will lose their half-hour a day TV privileges. In order to maximize the number of apps that they can share with each other they've decided that each kid will pick a different app. Of course, each kid likes different apps, and each wants the app they like best (or near best) to be the app on their phone. Here's how they've decided to select the apps: each kid writes down the apps they like on a list, starting with the app they like the most to the app they like the least (and leaving off apps they have absolutely no interest in). Then the oldest kid gets to pick the first app on his/her list to put on their phone. The second oldest gets to pick the first app on their list, unless it's already been picked by the oldest, in which case they get the second app on their list. This process repeats for each of the other kids – each gets the highest app on their list that hasn't already been picked. Given a list of each kid's app preferences, you must determine what app gets put on each phone.

Input

Input starts with a line containing a positive integer n ($n \leq 100$), where n is the number of kids. Following this are n lines, each starting with a positive integer m ($m \leq 50$) followed by m strings. The value of m indicates the number of apps on a kid's list, and the m strings are the names of the apps, from most preferred to least preferred. The first line is the preference list of the oldest kid, the second line is the preference list for the second oldest, and so on.

Output

Display the app selected for each kid's phone, from oldest kid to youngest, separating each app's name with a single blank. There will never be a test case where all of the apps on a kid's list will have been selected by older kids.

Sample Input 1	Sample Output 1
3 3 AppA AppB AppC 1 AppD 4 AppA AppE AppD AppF	AppA AppD AppE

Sample Input 2

```
4
5 SnapChart Rumbler Whimper Whine Tik
5 SnapChart Rumbler Whimper Whine Tik
5 SnapChart Rumbler Whimper Whine Tik
5 SnapChart Rumbler Whimper Whine Tik
```

Sample Output 2

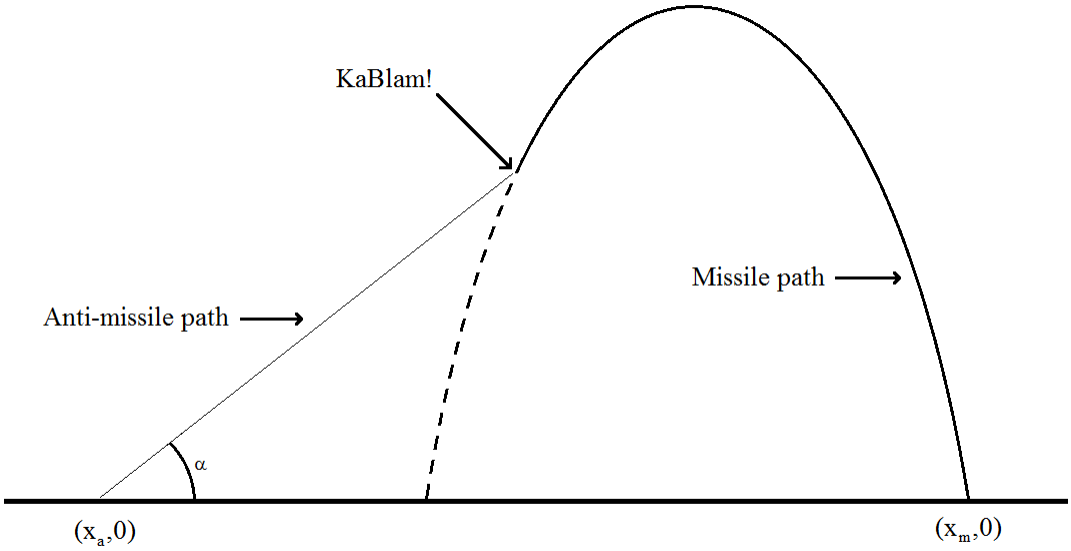
```
SnapChart Rumbler Whimper Whine
```

Problem D
KaBlam!

You've been asked to write software to help launch anti-missile rockets in order to take out missiles shot by the enemy. You've decided to use a simple 2-dimensional model, with the launch site of the missile located on the x axis at $(x_m, 0)$ and your anti-missile launch site at $(x_a, 0)$ (all values in feet). You also know the initial x and y velocities of the missile at the moment of launch: v_x and v_y (in feet per second). Given this information, you know that if the missile is launched at time 0 then the location of the missile at any later time t is given by

$$(x_m + v_x t, v_y t - 16t^2)$$

The image below shows one possible trajectory for the missile and the anti-missile. Note that the missile's trajectory is a parabola while the anti-missile's trajectory is a straight line. Also note in this scenario that v_x is negative (v_y will always be positive).



Your higher-ups want to be able to destroy the missile at a specific time t_K. Your job is to decide when to shoot your anti-missle and at what angle α so that you intercept the enemy's missile at that time. To aid in your calculations you also know the velocity of your anti-missile along its trajectory, v_a. Note that it might be impossible to destroy the missile at time t_K, either because you would have had to shoot your anti-missile before the enemy's missile is launched, or because the missile would have already landed (and blown up) prior to or at time t_K. If that's the case, your software should sound an alarm so that everyone can start running.

Input

Input consists of a single line containing six integers: x_m v_x v_y x_a v_a t_K, where $0 < v_y, v_a, t_K \leq 10000$, $-10000 \leq x_m, v_x, x_a \leq 10000$ and $x_m \neq x_a$.

Output

If it is impossible to destroy the missile at the requested time, display the phrase `start running`. Otherwise, display two values t_L α, where t_L is the time to launch your anti-missile and α is the launch angle (in degrees). Both values should have a maximum relative or absolute error of 10^{-4}.

Sample Input 1	Sample Output 1
`5000 -100 400 1000 500 20`	`14.87750061 38.65980825`

Sample Input 2	Sample Output 2
`5000 -100 400 1000 500 30`	`start running`

Problem E
Life Savings

Stanley is very prudent when it comes to his finances. He enjoys a somewhat spartan lifestyle and watches every penny he spends. Those few who call him friend view him as economical, abstemious, and parsimonious. He's basically a miserly tightwad.

Stan will never go shopping unless he has coupons. At the moment he has a bit of a conundrum. There are three items he wants to buy which have list costs of $100, $50 and $40. He has one coupon for 20% off his entire purchase which can't be used with any other coupon, and two coupons — one for 30% off any one item and one for 25% off any one item — which can be used together. He's not sure if using the first coupon by itself will save him more money than using the other two coupons together or vice versa, and the people standing in line behind him at the checkout counter are getting a little impatient. Can you help Stan with his problem and make cents out of this situation?

Input

Input consists of two lines. The first line starts with three positive integers p_1 p_2 p_3 ($p_1, p_2, p_3 \leq 1000$) which are the prices of three objects to be purchased. The second line contains three positive integers c_1 c_2 c_3 ($c_1, c_2, c_3 \leq 100$) where c_1 is the coupon percentage for the entire purchase (and cannot be used with any other coupon) and c_2 and c_3 are two one-item coupons (which can be used together but on separate items).

Output

If using the single entire-purchase coupon saves Stan more money, display the word `one` followed by the amount of savings; otherwise display the word `two` following by the best savings using the two one-item coupons. All savings should be output with two digits after the decimal point.

Sample Input 1	Sample Output 1
100 50 40 20 30 25	two 42.50

Sample Input 2	Sample Output 2
100 50 40 25 30 25	one 47.50

Sample Input 3	Sample Output 3
11 37 24 15 5 35	two 14.15

Problem F
Treating Fractions Irrationally

Anna has a big report to write concerning the voting habits of left-handed Democrats who own one or more kitchen blenders. She has a lot of statistics to report. For example, did you know that 56 out of every 87 left-handed Democrats who own one or more kitchen blenders can not name any of the current Supreme Court justices (yes, we were a bit shocked by this too.)

Now Anna is a bit irrational when it comes to fractions. She hates writing them as "56/87" but prefers the floating point version "0.64367816091954022..."

But this presents two problems. The first is that for most fractions, the floating point version will have to be truncated at some point. And this leads to the second problem: this truncated version is never equal to the original fraction. To minimize this second problem, Anna has decided that she will print the floating point version up to (but not including) the first 0 or 9 encountered, rounding down or up appropriately. So the above fraction would be written as "0.64367816", while the fraction 55/87 (which is equal to 0.6321839080459...) would be written as "0.632184". If the very first digit after the decimal point is 0 (as with the fraction 1/87) then Anna just writes "0"; if the first digit after the decimal point is 9 (as with the fraction 86/87) then Anna just writes "1". Finally if a 0 or 9 never appears in the floating point version (as with the fraction 58/87) then Anna just throws out that data (so the world will never know, for example, that 58 out of every 87 left-handed Democrats who own one or more kitchen blenders make great fruit smoothies).

Input

Input consists of a single line containing two positive integers n d, where $n < d \leq 100000$.

Output

Display Anna's truncated version of the fraction n/d. If the floating point version of n/d does not contain a 0 or 9, display the phrase `throw out`.

Sample Input 1	Sample Output 1
56 87	0.64367816

Sample Input 2	Sample Output 2
55 87	0.632184

Sample Input 3	Sample Output 3
58 87	throw out

Sample Input 4	Sample Output 4
1 87	0