ACM Spring 2019 Programming Contest

Lower Division

March 30th, 2019

Do not open until contest starts
Instructions for Participants

- Contest URL: `https://domjudge.cs.fsu.edu`
- You have 5 hours to answer questions.
- You may submit solutions in the following languages:
  - C/C++ (1999, 2011)
  - Java 8
  - Python (2.7 or 3.x)
  - Perl 5.22.1
  - Javascript (Node.js v4.2.6)
- You are only allowed access to official language documentation and COP3014 reference material. You are restricted to:
  - C/C++: http://www.cplusplus.com/reference/
  - Java: http://docs.oracle.com/javase/8/docs/api/
  - Python 2.7: https://docs.python.org/2/
  - Python 3.x: https://docs.python.org/3/
  - Perl: http://perldoc.perl.org/
  - Javascript: http://developer.mozilla.org/en-US/docs/Web/Javascript/Reference
  - COP3014 Reference:
    * http://www.cs.fsu.edu/~vastola/cop3014/
    * http://www.cs.fsu.edu/~jayarama/prog1.php
- You are also allowed one textbook or material no larger than 8.5" x 11" x 2" volume.
- No other resources (e.g. Stack Overflow, Google, Wikipedia) are permitted. Using non-permitted materials will lead to disqualification.

- Teams are restricted to using one workstation (computer) each.
- Use of a cell phone to circumvent these restrictions will lead to disqualification. Use of cell phones in contest rooms is not permitted.
- The Clarifications tab on Domjudge may be used to submit questions pertaining to each problem. Do not use this feature to request troubleshooting help.
- All input is redirected via STDIN.
- All output must be formatted to specification in terms of capitalization and spacing. Please refer to the example output for each question.
- Do not include a shebang in your submissions.
- **Scoring:**
  - Teams are ranked according to score. A higher score is rewarded by answering more questions while acquiring fewer penalties.
  - The team that solves the greatest number of questions in the quickest time wins.
  - Teams which solve the same number of problems are ranked by least total time.
  - Teams may resubmit solutions as many times as needed, but incorrect submission attempts will result in time penalties (and thus a lower score.)
  - The scoreboard may be accessed during the first four hours of the contest. The scoreboard will freeze during the final hour.

1 The Sandwich Factor

Tim, an aspiring computer scientist and math enthusiast, works the night shift at Myeraman's Sandwich Shoppe. One evening, during an extended lull in customer activity, Tim creates what he proudly calls a 'math sandwich'. His sandwich consists of two numbers, and a math operator. Being a self proclaimed mathlete, he states that to evaluate a math sandwich, an individual must take the given math operator and apply it to all integers between the two numbers which comprise the sandwich. Warning, these savory treats can be addictive.

1.1 Input

You will be given three lines of input in the format of:

$< integer >< singlespace >< integer >< singlespace >< operator >$

The second integer will always be greater than the first integer and there will always be at least one value between the two numbers (i.e. the numbers could be 7 and 9 but not 7 and 8). The operator after each number pair will either be addition, subtraction, or multiplication. If only one value exists between the two integers, then only output that one value.

1.2 Output

The task is to either add, subtract, or multiply all the integers between, but not including, the two integers given. For example the result of 9 13 + would be 33 i.e. 10 + 11 + 12. The output will be the result obtained by doing this for each of the three number pairs and their operators.

1.3 Sample Input/Output

Sample Input 1	Sample output 1
7 45 + 20 25 - 3 5 *	962 -48 4

2 Words are Numbers...right?

Last semester Tim took courses on reading, writing, and counting. This semester he has advanced in his mathematics courses and is now learning how to add. Sadly, he has failed his reading and writing classes, and must retake them along with his class on addition.

His addition course is a co-requisite to reading, so it assumes that he knows how to read somewhat. He has now been assigned a project that involves counting certain letters in words, and is really struggling, so it is your job to help this dingus complete his project.

2.1 Input

The input will begin with an integer **n**, followed by a newline. There will then be **n** strings all consisting of **n** characters, with a newline at the end of each string. The characters will all be in the range a-z. The smallest value for **n** will be 3.

2.2 Output

The output will a single integer that is the sum of the ascii values of each character in the diagonals formed by the strings if each string were stacked below the previous one as shown in the sample input.

Please note that when the diagonals intersect, you will be adding a letter twice. In the sample input below, the sum of the diagonals are (d + a + o) + (b + a + g), and a is added twice.

We have provided below the ascii values for a-z:

Letter	Value	Letter	Value
a	97	n	110
b	98	o	111
c	99	p	112
d	100	q	113
e	101	r	114
f	102	s	115
g	103	t	116
h	104	u	117
i	105	v	118
j	106	w	119
k	107	x	120
l	108	y	121
m	109	z	122

2.3 Sample Input/Output

Sample Input	Sample output
3 dog cat boo	606

3 Build-A-Pillow-Fort

After a long day of hard work, Tim likes to cozy up in a nice pillow fort. He's found that the best way to build the coziest pillow fort is to stack pillows up in a rectangular fashion, and then order each stack by its height. From left to right, each pillow stack will have a smaller height than the one to the right of it (or from the end of one row to the beginning of the next row).

To determine the "coziness" factor of a fort, Tim has devised a method. First, the pillow fort needs to be as cozy as possible, meaning it has to be sorted as described above. Then, you need to sum up the top and bottom rows and the leftmost and rightmost columns. Finally, the coziness factor will be the product of each of these sums. So, mathematically:

coziness = sum_top ∗ sum_bottom ∗ sum_leftmost ∗ sum_rightmost

3.1 Input

The first line of input will be the dimensions of the pillow fort, in the following format:

"m n"

without the quotes. **m** represents the number of rows, and **n** represents the number of columns. The next **m** lines will be the grid, each line having **n** numbers. Each number represents the height of each stack of pillows.

3.2 Output

The only output will be the product of the sums of each border (i.e. the coziness of the pillow fort) followed by a new line.

Remember you must sort the pillows before you calculate the coziness. This can be done by sorting the stacks in ascending order and placing them left to right, top to bottom, row by row.

3.3 Sample Input/Output

Sample Input	Sample output

3 3
10 12 8
20 50 1
3 5 7
	959400

4 Scheduling for Dummies

UH-OH! That dingus, Tim, has lost all of his appointments and we gotta help him get his affairs back in order! Luckily, he's pretty old fashioned, so we have a physical record of all of his appointments! We just don't know the order in which they are supposed to go!

Let's take a look at his records and help this mess of a man put his life back in order!

4.1 Input

Each of Tim's appointments are in a structured order. There will be an integer, **n,** on the very first line to denote how many appointments are on this record. **n** can be no larger than 100. The next **n** lines of input will be appointments structured in the following manner: Start Time, End Time, Day, Name.

- The times of day will be on a 24-hour clock, 0 being 12am and 23 being 11pm.
- The day will be a single integer between 1, being Monday, and 7, being Sunday.
- The name will always be a single word, no more than 10 letters long.
- There can be more than two appointments that are conflicting.

4.2 Output

The output of the program should print to standard out (cout) any conflicts in the order that they occur. The names should be printed in alphabetical order and separated by a forward slash. If there are no conflicts, simply print "NC".

4.3 Sample Input/Output

Sample Input 1	Sample output 1
4 12 13 1 Rick 12 13 1 Morty 12 13 2 Jerry 12 13 2 Summer	Morty/Rick Jerry/Summer

Sample Input 2	Sample output 2
3 1 2 1 Ricardo 12 15 2 Milos 6 7 3 Larry	NC

5 Royal Anagrams in the Lasagna Armory

King Timothy the Third has found himself in a peculiar situation. A dastardly denizen from a land far away has kidnapped his son, Prince Timothy the Sixth (yes, all his sons are named Timothy). The criminal has left a series of riddles describing where young Timothy could be, but King Timothy has no idea what any of it means.

That is where you come in. After several hours of examination, you realize a particular pattern about these riddles. They are filled with anagrams! You believe that hidden in all these anagrams is a message and after carefully evaluating the first note, you find this to be true. However, you cannot possibly do this for all the denizen's notes in one day. Luckily for you, you obtained a computer through totally unsuspicious means! You need to create a program that reads in the denizen's notes, finds the anagrams, and outputs the first most reoccurring character in the first word. Be careful, not everything has an anagram. In this case, output a "#". Good luck!

5.1 Input

The input file will hold a number, **n**, at the top indicating the amount of times you need to check for anagrams. Below this number will be n pairs of words for you to compare. Words are anagrams of each other if they use the same exact letters, and the same number of the letters. For example, help and pelh are anagrams, but helpl and pelh are not.

5.2 Output

If a pair of words are an anagram, output the character the occurs the most First, in the FIRST word. If they are not an anagram, output an "#". These should all be outputted on one line with no spacing in between characters and all characters should be printed upper case.

In the example below, in the original strings, helleo and oelleh are anagrams. The characters that occur the most are e and l, however l occurs twice before e occurs twice. As such, we would output L for that line of input.

5.3 Sample Input/Output

Sample Input	Sample output
3 helleo oelleh heptah ptaheh asf issisiss	LH#

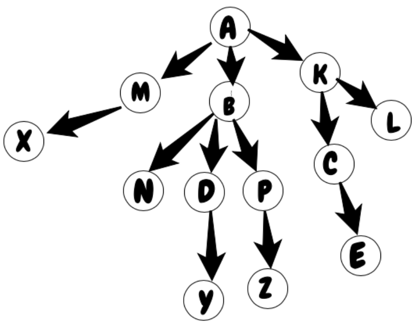
6 Office Battle Royale

Let us consider an office and you are given some relations among the employees based on their work positions. The relations given indicates a simple hierarchy of the positions like "who is working under whom".

For simplicity let us consider the employees are [A-Z]. And the relations among them are provided like the following.

A-M,A-B,A-K,B-D,P-Z,B-P,M-X,K-C,B-N,K-L,D-Y,C-E

The relations are comma separated and A-M means M is working under the supervision of A. Hence we can draw the following graph to represent all the relations provided above.



$$\begin{tikzcd}
 & & & A \arrow[dll] \arrow[d] \arrow[drr] & & \\
 & M \arrow[dl] & & B \arrow[dl] \arrow[d] \arrow[dr] & & K \arrow[d] \arrow[dr] \\
X & & N & D \arrow[d] & P \arrow[d] & C \arrow[d] & L \\
 & & & Y & Z & E &
\end{tikzcd}$$

Given the relations above, find the people with lowest work position/designation working under a given worker. In the example above, if given the worker B, the lowest work positions would be Y and Z.

6.1 Input

The relations along with the target person is provided in the same string. The format of the input string is A-B,B-C,B-D,...,A - the final letter having no relation added to it, is the target person for whom we want to find the subordinates.

6.2 Output

The output will be each worker with the lowest positions for the given target. The characters should be printed in upper casing, and be alphabetically sorted.

6.3 Sample Input/Output

Sample Input	Sample output
A-M,A-B,A-K,B-D,P-Z,B-P,M-X,K-C,B-N,K-L,D-Y,C-E,B	YZ