Computer Science Competition
State 2017
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	Arun
Problem 2	Changpu
Problem 3	Christina
Problem 4	Dai
Problem 5	Eric
Problem 6	Guang
Problem 7	Keerthi
Problem 8	Matthew
Problem 9	Rohan
Problem 10	Sasha
Problem 11	Yulia
Problem 12	Zachary

1. Arun

Program Name: Arun.java **Test Input File: None**

In a dream of going to the State UIL programming contest one afternoon as he was napping, Arun came up with the following interesting pattern. Can you replicate it?

Input: None

Output: Write a program that displays exactly the output you see below.

Exact Output Expected:

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
TSETNOCGNIMMARGORPECNEICSRETUPMOCLIUETATS
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
STATEUILCOMPUTERSCIENCEPROGRAMMINGCONTEST
```

2. Changpu

Program Name: Changpu.java **Test Input File: changpu.dat**

Changpu has just learned about numbers and their complements, especially since a complement is slightly different than the opposite of a number. He decides to write a simple program to test his newfound knowledge, but needs a little help. For fun, he wants to take any number, negative or non-negative, and print its complement if it is even, otherwise just print its opposite value if it is odd.

For example, the values 0 and -1 are complement values. So are 1 and -2, 2 and -3, 3 and -4, and so on. Opposites are different. The opposite of 1 is -1, 2's opposite is -2, and so on.

Input: A series of two-digit integers, all on one line, with single space separation.

Output: For each input integer N, if it is even, print N and its complement value, otherwise print N and its opposite value, with single space separation.

Sample Input:
```
83 -70 81 -13 51
```

Sample Output:
```
83 -83
-70 69
81 -81
-13 13
51 -51
```

3. Christina

Program Name: Christina.java **Test Input File: christina.dat**

In a recent computer science class, Christina has learned about string manipulation, and wants to use the names of her friends to practice her newly learned techniques. She decides to list all of her friends in a datafile, input them, and then use a special coding system to combine the names in certain situations to create some unique and interesting name combinations. For example, she'll use the letter 'U' to indicate uppercasing the entire name, 'L' to lowercase it, and 'R' to keep the name in its regular original format. Furthermore, she decides to reverse the order of the letters in the name in a similar fashion by using 'u', 'l', and 'r' which still performs the uppercase, lowercase, or keep regular casing, but reverses the order of the letters in the entire name.

She puts all of the names into a list, and then indicates which name to use with a 01 indicating the first name in the list, and 12 (if there are 12 names) to indicate the last name. In the sample data below, her name is in position 3 of the list of names, and so a code string of "03U" will produce her name in all uppercase letters, resulting in CHRISTINA. She then tries several codes, like "03u12R05L", which produces this result:

ANITSIRHCZacharyeric

The first portion of the code, "03u", causes her name, "Christina", to be shown in all uppercase letters, in reverse letter order. The next part, "12R", takes the 12th name in the list, "Zachary", and keeps it in regular initial case form, normal letter order. Finally, "05L" takes "Eric", and lowercases it, in normal letter order. All codes will consist of three characters, as shown in the examples here.

Input: An initial value N, followed by N names ($1 < N < 100$), all spelled with initial uppercase letters, followed by all lowercased letters. Following all of the names are several code strings which will produce interesting name combinations according to the rules stipulated above.

Output: The resulting name combination string produced by each of the code strings at the end of the data file.

Sample Input:

```
12
Arun
Changpu
Christina
Dai
Eric
Guang
Keerthi
Matthew
Rohan
Sasha
Yulia
Zachary
03u12R05L
01u05L06R
12R07U
10l03R09u
08R09u12l03u11l
12L05U
```

Sample Output:

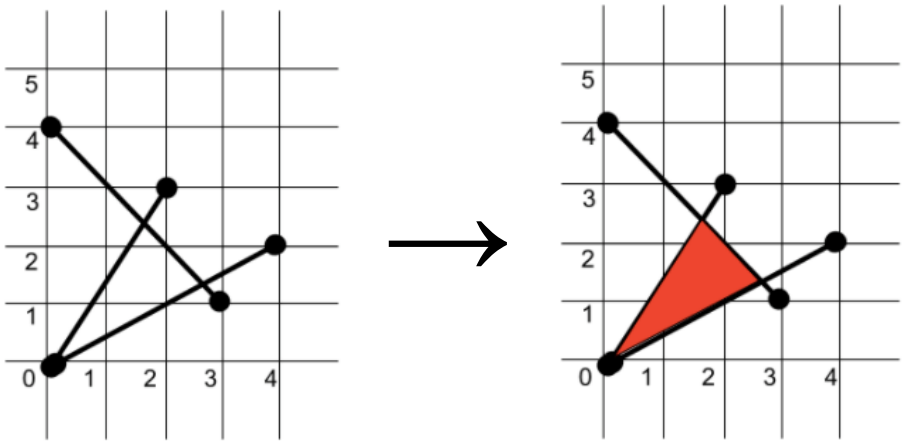
```
ANITSIRHCZacharyeric
NURAericGuang
ZacharyKEERTHI
ahsasChristinaNAHOR
MatthewNAHORyrahcazANITSIRHCailuy
zacharyERIC
```

4. Dai

Program Name: Dai.java **Test Input File: dai.dat**

Dai has a younger brother Dafydd, who likes to draw line segments, and has become very good at doing this. Dai wants to encourage this skill in his younger brother and is using his love of lines to teach Dafydd about polygons. Dafydd only draws straight line segments on a grid. On this grid, Dai determines if all the line segments collectively enclose a polygon, and if a polygon is enclosed, Dai tells Dafydd how many sides the polygon has.

For example, if Dafydd draws the following line segments, (0, 0) ⟶ (2, 3); (0, 4) ⟶ (3, 1); (0, 0) ⟶ (4, 2),



they enclose a polygon, which in this circumstance has three sides. However, Dafydd doesn't always draw line segments that enclose a polygon. Sometimes no segments intersect, and sometimes the intersecting segments do not enclose a polygon. When this happens, there is no polygon. Given Dafydd's lines, help Dai write a program to determine the number of sides of the polygon enclosed by the segments. If there is no polygon, display 0.

Input: An initial value N, followed by N lines of data. Each line contains a series of non-negative integers in groups of 4 values, the first pair representing the (x,y) coordinates of the start of a line segment, and the second pair the (x,y) coordinates of the end of that line segment.

Assumptions: A polygon has at least 3 sides. There will be no more than 1 closed shape per drawing. There will be at least one line on a drawing.

Output: The number of sides of the polygon enclosed by Dafydd's lines, otherwise output 0.

Sample Input:

```
3
0 0 2 3 0 4 3 1 0 0 4 2
0 0 2 2 3 3 5 4 15 15 14 3
1 6 4 9 3 9 6 7 6 7 7 4 2 4 8 5 1 6 4 3 19 9 18 8
```

Sample Output:

```
3
0
5
```

5. Eric

Program Name: Eric.java Test Input File: eric.dat

Number patterns have always fascinated Eric, and the latest pattern consists of the following process. He starts with two single digits in the range 0 through 9. The two digits can be the same value, or different values.

For example, he'll start with the values 2 and 6, and then will generate a third number that is the sum of the previous two, but if the sum exceeds 9, he only keeps the ones digit of that sum as the next value in the series. In this case, the sum of 2 and 6 is 8, which is the third number in the series. The next number is generated by taking the last two values, which are now 6 and 8, whose sum is 14, which exceeds 9, so he keeps only the 4. The next value is the sum of 8 and 4, which is 12, and so the 2 is the next value in the series. Next comes the sum of 4 and 2, which is 6. When the series matches the two values he started with, he stops and counts how many values were in the final series, which in this example would be 6: 2, 6, 8, 4, 2, and 6.

He realizes there probably isn't much use for this pattern, but finds it an interesting pattern exercise, just the same.

Input: Several pairs of single digit values in the range 0-9, with single space separation. A pair can be matching values.

Output: For each pair, output the number of values that result in the series at the point that the beginning pair of value is matched by the last pair of values.

Sample Input:

```
2 6
1 3
5 5
```

Sample Output:

```
6
14
5
```

6. Guang

Program Name: Guang.java **Test Input File: guang.dat**

Numbers have always fascinated Guang, and recently he has been playing around with the idea of expressing numbers in different bases. To make it even more interesting, he adds zeroes to the front of the converted value equal to how many digits are in the converted value.

For example, the base 10 value 22 converts to 1333 in base 3, and so he writes his final value with four leading zeroes, 00001333. The decimal value 17 becomes 0021 in octal using his output scenario, and the value 29 becomes 001D in base 16.

Input: Several pairs of integers N ($0 < N < 10000$) and B ($1 < B < 37$), each on one line with single space separation, N representing a base 10 value, and B representing the base to which Guang will convert the value N.

Output: The converted value according to the expression format described above, with as many leading zeroes as there are significant digits in the newly converted value.

Sample input:

```
22 3
17 8
29 16
```

Sample output:

```
000211
0021
001D
```

7. Keerthi

Program Name: Keerthi.java **Test Input File: keerthi.dat**

Keerthi has discovered a treasure map! However, in the treasure map some of the land mass runs off the edge which means it is part of the continental land mass and not a part of any island. She knows the treasure is on an island, but she needs help identifying how many islands there are, so she knows how many places to look.

By definition, an island is surrounded by water, so if land runs off the edge of the map, then it is not an island. All of Keerthi's maps are represented by a 10x10 grid of characters. Water is represented by '.' and land by '#'. Given this map, count the number of islands.

Input: An initial value N, followed by N data sets. Each data set is a 10x10 grid of '.' for water and '#' for land. Each data set will be followed by a line containing a single '-'.

Output: The number of islands on the treasure map.

Sample Input:

```
3
.......###
..#......#
..........
....###...
..#######.
...#####..
.....#....
.....#..##
.##.....##
........##
-
.......###
.....#####
......####
....###...
..#######.
...#####..
.....#....
.....#####
.##.....##
.##.....##
-
.....#....
..#......#
......####
.##.......
......##..
...#####..
.#...###..
.....#....
.#####....
..........
-
```

Sample Output:

```
3
0
4
```

8. Matthew

Program Name: Matthew.java **Test Input File: matthew.dat**

Searching for words using wild cards has been a challenge for Matthew in his algorithms class and he needs your help. He knows that the '*' character means to look for any character as many times as possible, even zero times. He also knows that the '?' means to look for any single character, once and only once, and that those two wild card search characters can be embedded in a some kind of search string.

For example, if he has a list of words like:

BILLS BELLS TILLS DOLLS DOLLIES 2BILLS 5BELLS

the search string **"B*"** will match any word that starts with the letter "B", including just the letter "B". With the above list of words, this search would result in the words "BILLS", "BELLS", and "B".

Using "D?LLS" will match only match the word "DOLLS", since that is the only one in this list of words that starts with "D", ends in "LLS", and contains a single letter in between.

In addition to using these two wildcard characters, he understands that a range of characters can be included in a search string, using the "[]" symbols. For example, the search string "[2-5]B?LLS" will look for a single instance among the digits 2, 3, 4 or 5, followed by a "B", then a single character, and ending with "LLS".

It is also possible that his search comes up empty, which means no words in the list match the search string. In those cases, he reports an output of "NONE".

Input: Several data sets, each of which begin with a list of N words (1 < N < 20) consisting of uppercase letters and digits, all on one line with single space separation. The line following the word list contains a single integer P (1 <= P <= 20), followed by P search strings containing the search string components described above in various combinations.

Output: For each search string, output the resulting list of words that are matched by the string, or the word NONE if there are no matches.

Sample input:

```
BILLS BELLS B TILLS DOLLS DOLLIES 2BILLS 5BELLS
3
B*
D?LLS
[2-5]B?LLS
UIL DISTRICT REGION STATE CONTEST COMPUTER SCIENCE PROGRAMMING CHAMPION
4
C*
*?ON
?[I-M]*
*MM*
```

Sample output:

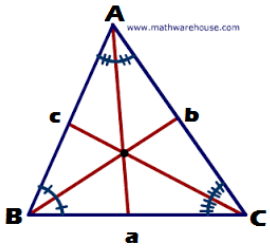
```
BILLS BELLS B
DOLLS
2BILLS 5BELLS
CONTEST COMPUTER CHAMPION
REGION CHAMPION
UIL DISTRICT
PROGRAMMING
```

9. Rohan

Program Name: Rohan.java **Test Input File: rohan.dat**

In a recent geometry lesson on triangles and the various aspects they possess, Rohan needs help finding two interesting points – the incenter and the centroid. He also needs to find the area of any triangle, but he already knows Heron's formula and does not need help with that, which he remembers uses the semi-perimeter of the triangle in an interesting formula that takes the square root of the product of four things: the semi-perimeter itself, and the three differences between the semi-perimeter and each of the three sides a, b, and c. He also realizes he needs to use the distance formula to find the lengths of the sides a, b and c, but already knows that one too.

He decides to look up what those **incenter** and **centroid** mean, and discovers that an **incenter** is a point in the middle of any triangle where all three **angle bisectors** meet, and that the **centroid** is another point in the middle of the triangle where all the **medians** of a triangle meet. Below are diagrams illustrating these two points.



Incenter Centroid

In an online search, he finds formulas for these two interesting points, not fully comprehending how they work, but figures if he can code them correctly, he'll be OK.

The incenter formula for the Y value (YI) is a bit more complex:

YI = (a*Ya + b*Yb + c*Yc) / (a+b+c)

with a similar formula used for XI.

The centroid X value (XC) uses a formula that averages the three X coordinates (Xa, Xb, Xc) of the vertices of the triangle: **XC = (Xa + Xb + Xc)/3**
with the YC value using a similar formula.

Input: Several sets of six real number values no farther from zero than a distance of 100, representing the X and Y coordinates of three vertices of a triangle A, B, and C, in the order Xa, Ya, Xb, Yb, Xc, Yc.

Output: Two sets of coordinates representing the centroid and incenter of a triangle, and a third value representing the area, in the format shown below, with the two coordinate pairs included inside parentheses, separated by a comma with no spaces, and the area as a single real value. All values are to be expressed using rounded precision to the hundredths place.

Sample Input:

```
-2 4 4 2 0 -6
-2 7 7 4 1 -2
```

Sample Output:

```
(0.90,0.72)
(0.67,0.00)
28.00
(2.15,2.85)
(2.00,3.00)
36.00
```

10. Sasha

Program Name: Sasha.java **Test Input File: sasha.dat**

Sasha enjoys painting. He has been creating many wonderful pieces of art, so beautiful that the internet keeps stealing his artwork! He wants your help in identifying his artwork across the internet so it can be flagged to be taken down from the unauthorized sources. However, as people copy and share his art, the background tends to get cropped and the image keeps getting scaled to different sizes. This makes it harder for Sasha to automatically detect if an image is of his artwork or not.

Given two images, the first of which is Sasha's original work, and the second of which is an image from the web, determine if they are the same or different. All of Sasha's images consist of abstract shapes drawn with `'#'` and a background drawn with `'.'`. In the images across the web, sometimes the background is cropped a little different and the image has been scaled up or down by some amount. Sasha can tell if the web image is his original work if, when scaled and the background removed, the two abstract shapes are the same. Sasha's original work and the images on the web can be of any size, but they are always square in shape.

Input: An initial value N, followed by N data sets. Each data set has two square images, followed by two dashes: `"--"`. Each image is separated by a single dash: `'-'`. Each image is square in shape, made up of `'#'` and `'.'` characters. The first image of each pair is Sasha's original art work. The second image is the one found on the web.

Output: The word `"SAME"` if the two images contain the same abstract shape once scaled or translated, or `"DIFFERENT"` if they do not share the same abstract shape.

Sample Input:

```
3
.....
..##.
..#..
.....
.....
-
####..
####..
##....
##....
......
......
--
.....
.###.
.###.
.....
.....
-
....
.#..
##..
.#..
--
.....
.####
.####
.####
.####
-
..........
#########.
#########.
#########.
#########.
#########.
#########.
#########.
#########.
#########.
--
```

Sample Output:

```
SAME
DIFFERENT
SAME
```

11. Yulia

Program Name: Yulia.java Test Input File: yulia.dat

Yulia enjoys word searches, but she has decided to crank it up a notch. In word searches, given a word bank, she tries to find words in vertical or horizontal directions. In a new word search, given a word, she aims to find which words in the word search most closely match the word. A word in the word search can closely match the given word if it is missing some letters or has some extra letters.

For example, suppose Yulia is looking for the word "hatelice" in the following word search:

```
chAlice
bbbbbbh
ccccccl
ddddddi
eeeeeeC
fFfffe
Ggggggn
```

The word “hatelice” never shows up verbatim in this word search. However, there are two words that are close. The word "chalice" and "ehlicen". The word “chalice” is close to the word “hatelice” because the “c” needs to be added in the front of the word, and the “te” needs to be removed: “chatelice”. The number of unaltered letters is 6. When we do the same for "ehlicen", we get a similar result: “ehatelicen”. In this case, the number of unaltered letters is 5. Therefore, “chalice” is closer to “hatelice” than “ehlicen”.

For each word search and the given word, output the number of unaltered letters in the closest word to the given word. Please note that Yulia only looks for words vertically and horizontally, and she always ignores case.

Input: An initial value N, followed by N data sets. Each data set will begin with an integer X, which represents the size of the word search to follow. Each of the following X lines will have X characters representing the word search. The line following the word search will contain Yulia's search word.

Output: The number of unaltered letters of the closest word in the word search to the search word.

Sample Input:

```
3
7
chAlice
bbbbbbh
ccccccl
ddddddi
eeeeeeC
fFfffe
Ggggggn
hatelice
5
JAZZY
FUZIL
TIZZY
MUJIK
UEWYY
spazzy
6
ATATQT
TRTATA
AMPTAT
TATATN
DTATAT
TATETA
ATATATATATATAT
```

Sample Output:

```
6
4
5
```

12. Zachary

Program Name: Zachary.java **Test Input File: zachary.dat**

Zachary has always had fun generating number patterns, and his latest idea is to create a box of numbers that represent the one's place of a series of products of two values. For example, he takes the two values 7 and 9 and makes a box with 7 rows and 9 columns, with the products of the values from 1 to 7 and 1 to 9, as shown below.

```
1 2 3 4 5 6 7 8 9
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
```

The first row contains the products of 1 and the values from 1 to 9.
Numbers in the second row are products of 2 and the values from 1 to 9.
The third-row values are products of 3 and the values from 1 to 9.
This continues until reaching the seventh row, which contains products of 7 and the values 1 to 9.

Once he has these values, he only takes the one value of each product, jams them altogether into a box, which makes an interesting pattern that fascinates him. He calls this pattern a "one's box".

```
123456789
246802468
369258147
482604826
505050505
628406284
741852963
```

Input: A series of pairs of single digit positive integers, each pair on one line, separated by a single space.
Output: Create and output a "one's box" with each pair of integers, according to the description and example shown above.

Sample Input:

```
7 9
5 4
2 3
```

Sample Output:

```
123456789
246802468
369258147
482604826
505050505
628406284
741852963
1234
2468
3692
4826
5050
123
246
```