ECOO 2012
Programming Contest
Questions

Local Competition (Round 1)

After March 18, 2012

Problem 1: Sluggers

Two important stats in baseball are the team batting average and the team slugging average. Batting average is defined as the total number of hits (this includes 1 base hits, 2 base hits, 3 base hits and home runs combined) divided by the total number of times at bat ("at bats") for all players on the team. The team slugging average is defined using the following equation:

$$sa = \frac{A + 2 \times B + 3 \times C + 4 \times D}{E}$$

Where A is the number of 1 base hits, B is 2 base hits, C is 3 base hits, D is home runs, and E is the number of at bats for all players on the team. Both slugging and batting averages are always presented as decimals rounded to 3 places, leaving off the leading 0 (in theory batting averages can be as high as 1.000, and slugging averages as high as 4.000 but in practice they are both usually well below 1).

DATA11.txt (DATA12.txt for the second try) contains the raw data on the top 10 teams during a regular season of Major League Baseball. The first line is the season name, followed by 10 lines for each of the top 10 teams. Each of these lines starts with a team name (single word) followed by 7 integers: Games Played, At Bats, Runs, Hits (total), two-base hits, three-base hits, and home runs. One space character separates each item on each line.

Write a program to produce a report showing the batting and slugging averages for each team in the order they appeared in the input file, and formatted EXACTLY as shown below, including all punctuation and matching upper and lower case exactly. All spacing is done with a single space character. All batting and slugging averages will be less than 1. The final line shows the same averages for all 10 teams combined (computed from the sum of all at bats, hits, etc. for all 10 teams). Note that the two lines of "=" characters each contain 20 characters.

Sample Input

```
2011 Regular Season
Boston 162 5710 875 1600 352 35 203
NY_Yankees 162 5518 867 1452 267 33 222
Texas 162 5659 855 1599 310 32 210
Detroit 162 5563 787 1540 297 34 169
St.Louis 162 5532 762 1513 308 22 162
Toronto 162 5559 743 1384 285 34 186
Cincinnati 162 5612 735 1438 264 19 183
Colorado 162 5544 735 1429 274 40 163
Arizona 162 5421 731 1357 293 37 172
Kansas_City 162 5672 730 1560 325 41 129
```

Sample Output

```
2011 Regular Season
====================
Boston: .280 .461
NY_Yankees: .263 .444
Texas: .283 .460
Detroit: .277 .434
St.Louis: .273 .425
Toronto: .249 .413
Cincinnati: .256 .408
Colorado: .258 .410
Arizona: .250 .413
Kansas_City: .275 .415
====================
Big 10 Av: .267 .428
```

Problem 2: Decoding DNA

DNA is made up of two twisted strands that encode genes using long combinations of four bases: Adenine, Cytosine, Guanine and Thymine. The strands are complementary to one another, meaning that Adenine and Thymine are always opposite each other, and Cytosine and Guanine are always opposite each other, like this.

A double strand of DNA: ATCAAGGCCTATTCCGGGAAAGG
TAGTTCCGGATAAGGCCCTTTCC

In order for the information in a gene to be used, it has to be transcribed into a strand of RNA. During this process, a portion of one strand of DNA is transcribed – this portion is known as the transcription unit. The start of the sequence to be transcribed is signaled by a sequence of bases known as a promoter, and the end is signaled by a sequence known as the terminator. For our purposes, the promoter is the sequence TATAAT, which begins 10 bases before the start of the transcription unit, and the terminator consists of two distinct, complementary, reversed sequences of at least length 6 that cause the RNA molecule to coil back on itself and pinch off the transcribed strand. If TATAAT appears twice on a strand, only the first occurrence counts as the promoter. An example is shown below.

AGATTA**TATAAT**GATAGGATTTAGATTGACCC**GTCATGCA**AGTCCA**TGCATGAC**AGC

In this example the promoter and terminator sequences are boldfaced, and the transcription unit is underlined. The resulting RNA will be complementary to the transcription unit, except that in RNA Uracil takes the place of Thymine. For this example, the result looks like this:

CCUAAAUCUAACUGGG

DATA21.txt (DATA22.txt for the second try) will contain five single strands of DNA, one on each line. Write a program to output the RNA sequence that results from the transcription process. The sequences should be numbered starting at 1, with a colon and a single space character following the number, as shown below.

Sample Input

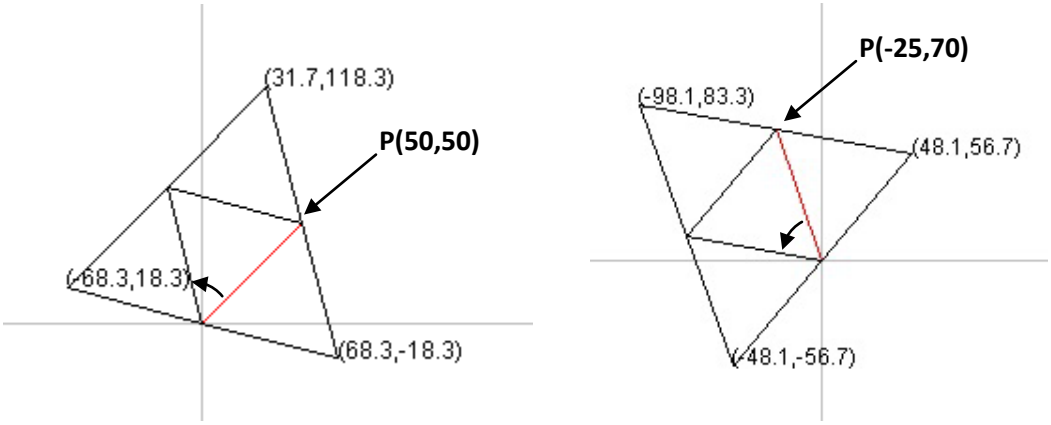
```
AGATTATATAATGATAGGATTTAGATTGACCCGTCATGCAAGTCCATGCATGACAGC
AGTCTTCAAGGGGATTCCCAGGTATATAATGCAGATCGCGACGAAATATCGGGCGGGATCCATACCGACCCAGCCGCCCGA
TATAATGGGGGAGAGACCGAGTGTTTAAGTCCCGAGGGATCGGGAGTGAGATTGAGGGAATTCCGGGAATCTCACT
```

Sample Output

```
1: CCUAAAUCUAACUGGG
2: UAGCGCUGCUUUAU
3: CUCUCUGGCUCACAAAUUC
```

Problem 3: Triangles

Who doesn't love a good equilateral triangle? Equilateral triangles have a lot of symmetry – so much so that if you fix one point of an equilateral triangle to the origin, then all you need is one other corner point to begin generating a tiling of identical equilateral triangles. In the examples below, the points labeled P were given as input, and the large equilateral triangles, each made up of 4 smaller, identically-sized ones, were generated automatically. Note that in these tilings, the third point of the innermost triangle is always counter-clockwise from the given point P, as shown by the arrow.



DATA31.txt (DATA32.txt for the second try) will contain 5 test cases. Each test case consists of two integers $\mathbf{P_x}$ and $\mathbf{P_y}$, the x and y coordinates of point **P**. Your job is to compute the coordinates of the three corners of the large equilateral triangle constructed as described above. These coordinates should be rounded to one decimal place and reported on a single line. Each point should be bracketed and spaced exactly as shown, and there should be a space separating each point on the line. The order in which you report the three points on the line does not matter.

Sample Input

```
50 50
-25 70
-35 -23
35 -23
5 6
```

Sample Output

```
(-68.3,18.3) (68.3,-18.3) (31.7,118.3)
(-48.1,-56.7) (48.1,56.7) (-98.1,83.3)
(37.4,-18.8) (-37.4,18.8) (-32.6,-64.8)
(2.4,41.8) (-2.4,-41.8) (72.4,-4.2)
(-7.7,1.3) (7.7,-1.3) (2.3,13.3)
```

Problem 4: Boggled

In the game of Boggled, players are given a 4x4 array of letters and about 2 minutes to list as many words as they can find in the array of letters. A word only counts if it is 3 or more letters long and you can trace at least one path through the letters to spell the word without using any letters twice.

In the example at left (depicting the offline version of the game) the two styles of arrows trace the words SALT and MACE.

For the purposes of scoring, 3- and 4-letter words are worth 1 point, 5-letter words are worth 2 points, 6-letter words are worth 3 points, 7-letter words are worth 4 points, and anything longer is worth 11 points. Paths can travel in any direction, including diagonally. There is no penalty for listing words that are too short or not present on the board and there is no penalty for listing the same word twice, but none of these are worth any points. (Of course in the real game, words would also have to be found in some standard dictionary, but we're not going to worry about that here.)

DATA41.txt (DATA42.txt for the second try) will contain 5 test cases. The first 4 lines of each test case will be a Boggled board, followed by an integer n on a line by itself, and then a list of n words that represent a single player's turn. Your program must output that player's score along with the number of legal and illegal words in the list. Illegal words are either not found on the board, too short, or repetitions of words higher up in the list. If a word is illegal for more than one reason, it is recorded once in the most important category of illegal words, with too short being most important, followed by repetition, and then not found. The format of the output must be exactly as shown below.

Sample Input

```
HIWH
PXBY
ASQK
PVES
15
SAX
SAP
WHIP
WHY
PAP
PAX
VASE
VASES
BY
HIP
HIPS
SAVE
SAVES
PAVE
PAVES
LSUN
DAGE
EBUI
WERJ
37
SUN
SUNG
LANGUAGE
SAGE
BAG
SAG
BEE
WED
WEDS
BED
BEDS
BALD
BALDS
BUG
BUGS
LAGS
LAG
SAD
BEAD
BEADS
EAD
EADS
LAD
LADS
GUN
GAS
BAG
BAGS
BAD
BADE
JIG
JIGS
JIGSAW
RIG
RIGS
RUG
RUGS
```

Sample Output

```
Your score: 16 (13 good, 1 not found, 1 too short, 0 repeated)
Your score: 36 (34 good, 2 not found, 0 too short, 1 repeated)
```

ECOO 2012
Programming Contest
Questions

Regional Competition (Round 2)

April 28, 2012

Problem 1: Prime Time

Alice is sending Bob coded messages using an encoding scheme of her own design. When she wants to send a message, she starts by turning the characters of the message into integers. This is done by assigning 0 to the space character, then 1 to A, 2 to B, and so on through the alphabet. She also assigns 27 to the period character, 28 to the comma, 29 to the exclamation mark, and 30 to the question mark.

After Alice converts characters to numbers, she puts each pair of numbers together to make a new 4-digit number (if she needs to, she pads the message with an extra space at the end). Then she selects a six-digit prime number less than 500,000 as the "key" value. She multiplies each of the 4-digit numbers in the message by the key and transmits the resulting numbers, starting with an un-encoded integer that indicates how many coded numbers are coming.

Example:

"HEY BOB!" → "HE" + "Y " + "BO" + "B!" → 0805 2500 0215 0229

Key: 395611
Encoded Message: 4 318466855 989027500 85056365 90594919

The genius part of this scheme is that when Alice sends Bob a message, Bob doesn't need to know the key ahead of time. So she is free to choose any key she wants and she can change the key for each message. Unfortunately, the "genius" part also makes her coded messages very easy for a third party to crack.

DATA11.txt (DATA12.txt for the second try) contains five coded messages from Alice to Bob (minimum length 4 characters per message). Your job is to decode each message and display the results, one message per line.

Sample Input

```
4 318466855 989027500 85056365 90594919
6 904474779 361632680 1100621200 907226332 47169480 1179237000
4 307114756 570239600 307725727 549873900
8 149471983 450198915 810358047 802334700 149471983 450198915 810358047 802334700
7 105593497 195549029 74466500 14893300 32467394 74615433 168145357
```

Sample Output

```
HEY BOB!
WAIT, WHAT?
OH, OK.
ECOO... ECOO...
GIMME A BREAK!
```

Problem 2: Password Strength

Passwords are the most basic information protection devices. But many people compromise the security of their data by choosing passwords that are easy for a human or a bot to guess. The table below summarizes one set of criteria for determining the strength of a password, and shows how to apply these criteria to two different example passwords.

Example 1: "ECOO()123abcd9876" **Example 2:** "1234512345"

Score	Description	Ex 1	Ex 2
Length	Score 4 points for each character in the password.	+68	+40
Basic Requirements	To qualify, must be at least 8 chars long and also contain 3 of the four basic character types (upper case letters, lower case letters, digits, and symbols*). Score two points for the length plus another two for each of the four types of characters it contains.	+10	--
Upper Case	Add (length – n) * 2, where n is the number of upper case letters and n > 0.	+26	--
Lower Case	Add (length – n) * 2, where n is the number of lower case letters and n > 0.	+26	--
Digits	Add 4 * n, where n is the number of digits, but only if n < length.	+28	--
Symbols	Add 6 * n, where n is the number of symbol characters.	+12	--
Middle Digits and Symbols	Add 2 * n, where n is the number of digits and symbols not in the first or last position.	+16	+16
Letters Only	If the password contains only letters, subtract 1 point for each letter.	--	--
Digits Only	If the password contains only digits, subtract 1 point for each digit.	--	-10
Consecutive Upper Case	Subtract 2 * (n – 1) for each set of n consecutive upper case characters.	-6	--
Consecutive Lower Case	Subtract 2 * (n – 1) for each set of n consecutive lower case characters.	-6	--
Consecutive Digits	Subtract 2 * (n – 1) for each set of n consecutive digits.	-10	-18
Sequential Letters	Subtract 3 * (n – 2), where n is the length of the longest case-sensitive sequence of letters longer than 2 (e.g. "abcd" or "CBA" but not "aBcD" or "SJD" or "a").	-6	--
Sequential Digits	Subtract 3 * (n – 2), where n is the length of the longest sequence of digits longer than 2 (e.g. "9876" but not "28" or "6").	-6	-9
TOTAL SCORE	Add up all the above. Negative scores become 0. Scores over 100 become 100. 0-20 = Very Weak, 21-40 = Weak, 41-60 = Good, 61-80 = Strong, 81-100 = Very Strong	100 Very Strong	19 Very Weak

*A symbol is any character that is not a letter or digit.

DATA21.txt (DATA22.txt for the second try) contains ten passwords, one per line. Write a program that reads each password, computes its total score based on the above criteria, and categorizes the password as Very Weak, Weak, Good, Strong, or Very Strong.

Sample Input

ECOO()123abcd9876
1234512345
ecoo2012

Sample Output

```
Very Strong (score = 100)
Very Weak (score = 19)
Good (score = 47)
```

Problem 3: Airport Radar

A passenger jet leaves from John C. Munro International Airport in Hamilton, Ontario and flies on a straight path until it reaches its destination. When it begins its flight, it will be seen by the radar tower located at Munro Airport. Along its path, it will pass by other airports, each with their own radar towers. In some cases, the airplane will pass close enough to be seen, in others it won't. Your task is to determine how many radar screens the airplane will appear on during its flight.

DATA31.txt (DATA32.txt for the second try) will contain five sets of data. The first line of each set contains integers D, L and N. Integer D corresponds to the airplane's direction of travel in degrees relative to due East (East = 0, North = 90, West = 180, etc.). Integer L corresponds to the length of the flight (that is, the distance travelled). Integer N corresponds to the number of airports in the area, excluding Munro Airport. The next N lines each contain 3 integers X, Y, and R giving the coordinates of the airport (X, Y) and the range of its radar tower (R). Coordinates are relative to Munro Airport, which is assumed to be at location (0, 0). The X axis increases to the East and the Y axis increases to the North. All units are in kilometers, and you should assume that a radar tower can pick up any object that is at most R km away.

Write a program that will read each test case and print out the number of radar towers that will see the jet at some point during its flight.

Sample Input

```
45 234 3
100 100 25
-100 0 150
250 300 70
120 400 5
100 100 25
-100 120 50
-230 270 70
-250 -250 200
-200 345 10
```

Sample Output

```
The jet will appear on 3 radar screens.
The jet will appear on 4 radar screens.
```

Problem 4: Lo Shudoku

Jenny is obsessed with 3x3 Magic Squares. In case you have forgotten, a 3x3 Magic Square contains all integers from 1 to 9 arranged so that every row, column, and diagonal adds up to the same number. The earliest known Magic Square is the Lo Shu Square (shown at right) discovered in China more than 2600 years ago. There are only 7 other 3x3 Magic Squares, all obtained by rotation and reflection of the original Lo Shu Square:

4	9	2
3	5	7
8	1	6

8	1	6
3	5	7
4	9	2

6	1	8
7	5	3
2	9	4

4	3	8
9	5	1
2	7	6

2	7	6
9	5	1
4	3	8

2	9	4
7	5	3
6	1	8

6	7	2
1	5	9
8	3	4

8	3	4
1	5	9
6	7	2

Jenny's friends have learned not to leave their Sudoku puzzles unattended when she's around. She will take any fully or partially completed Sudoku puzzle she finds and play a game of her own invention called "Lo Shudoku", in which she transforms the 9x9 Sudoku board into a set of nine Magic Squares using a restricted set of moves. Then she writes down a "Lo Shudoku" square in the margin that shows the number of moves she used for each of the nine squares:

Original Sudoku Board

	1		5	9	6	8	7	3
9			4	1	3			6
		5	8	7	2	4	1	9
1	7	2		4			9	
8	6	3	2	5	9			
5	4	9		6			3	8
7	2	6	9	8	4	3	5	1
3			1					4
4	9	1	6	3	5	7	8	2

Jenny's Transformation

8	1	6	4	9	2	4	9	2
3	5	7	3	5	7	3	5	7
4	9	2	8	1	6	8	1	6
2	9	4	8	3	4	2	9	4
7	5	3	1	5	9	7	5	3
6	1	8	6	7	2	6	1	8
8	1	6	8	3	4	8	1	6
3	5	7	1	5	9	3	5	7
4	9	2	6	7	2	4	9	2

Jenny's "Lo Shudoku" Square

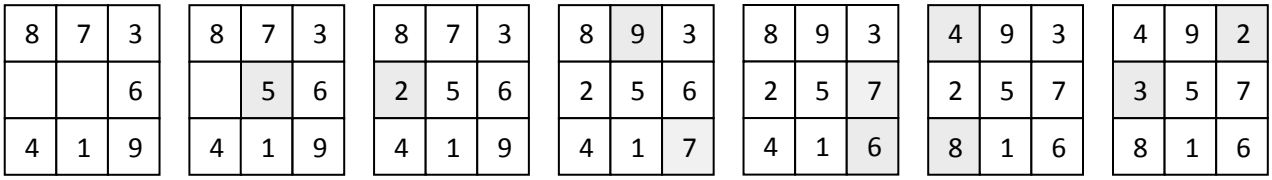
8	5	6
5	7	7
4	6	7

When she plays Lo Shudoku, Jenny works on each of the nine 3x3 squares, turning them into Magic Squares one by one. When transforming a 3x3 square, she always uses as few moves as possible, and she always proceeds in two separate phases:

1. First, she fills in all the blank spaces with the remaining integers from 1 to 9. Each number filled in counts as one move.
2. Once all the blanks are filled, she repeatedly swaps pairs of integers until she has a Magic Square. Each swap counts as one move.

When she has transformed each square in this way, she produces her Lo Shudoku square showing the number of moves she used to create each of the nine Magic Squares. In the example above, the Lo Shudoku square shows that the top left Magic Square took 8 moves, the ones to its right and beneath it took 5 moves each, and so on.

Here is one way that Jenny could transform top right 3x3 square from the example above:



In phase one she fills in the 5 and 2 first, then in phase two she makes 4 swaps to complete the Magic Square. This is not the only way she could have made a Magic Square from the original numbers, but there is no shorter path to a Magic Square than this. She records this minimum number of moves (6) in the top right corner of her Lo Shudoku square.

DATA41.txt (DATA42.txt for the second try) contains 5 test cases. Each test case will consist of 9 lines of 9 digits, representing a Sudoku board. The Sudoku board might be completely filled in, or it might be partially filled in. If it is partially filled in, the blank spaces will be represented as zeros. Your program must output a Lo Shudoku square for each Sudoku board given. For ease of reading, each Lo Shudoku square should be followed by "---".

Sample Input

```
010596873
900413006
005872419
172040090
863259000
549060038
726984351
300100004
491635782
329847651
748561392
156392478
```

```
514726839
673985124
892413765
965138247
287654913
431279586
206004900
040900100
900800002
700045090
064389270
090120004
```

```
400003008
005008020
003500407
093006500
652830070
007500036
000060108
000908000
501040000
140002600
060097315
009600280
```

```
000050086
649300070
000640100
030021009
106000708
400980030
008072000
060003852
250060000
```

Sample Output

```
856
577
467
---
655
443
525
---
```

```
787
878
888
---
688
988
977
---
```

```
888
787
778
---
```

ECOO 2012 Programming Contest

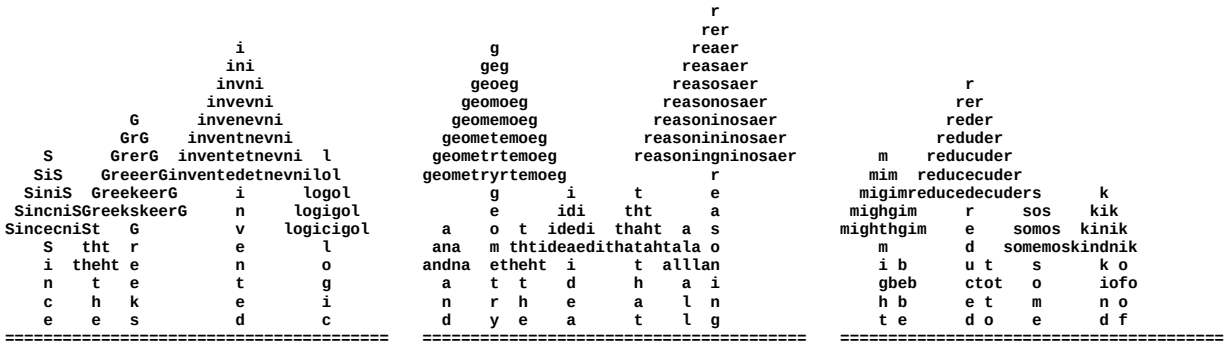
Final Competition (Round 3)

May 12, 2012

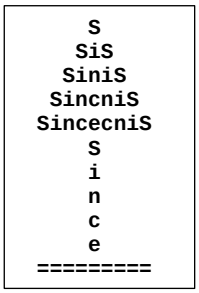
Problem 1: The Word Garden

In addition to creating problems for the ECOO competition, I have an unusual landscaping business on the side. I create custom-made “Word Gardens”. To make a Word Garden, I take one of my customer’s favorite quotations, turn each word in the quotation into a tree, and then plant the trees in rows, packing them as tightly together as I can.

Here’s an example, using a paraphrased quote from Hubert Dreyfus’ classic book, *What Computers Can’t Do*. The original quote was, “Since the Greeks invented logic and geometry the idea that all reasoning might be reduced to some kind of calculation has fascinated most of the West's rigorous thinkers”.



Notice that each word tree has a “trunk” consisting of the original word repeated twice vertically, and then “leaves” added to the top half of the trunk so that each level of leaves corresponds to an increasing portion of the word, repeated forwards and backwards and centered on the trunk. The tree for the word “Since” is shown at right.



My word trees are always planted in rows with an exact width of 40 characters I plant the trees from left to right, placing each one as far to the left as I can without overlapping any previously planted tree. The ground consists of 40 “=” characters, and the maximum height of any tree is 23 characters (not including the ground).

DATA11.txt (DATA12.txt for the second try) will contain 5 lines of text. Each line of text will consist of a list of words with each pair of words separated by a single space character. Each word will consist of at least 1 and no more than 23 characters (expect alphanumeric characters and also punctuation characters in some cases).

Each line of text should be turned into a line of trees, as described above. The ground should always consist of exactly 40 "=" characters. Each line of trees will always fit into a space of 24 rows by 40 columns. You should make sure you are using a fixed-width font (like courier) or the output will not look right.

Because of the size of the output, you may use input control to show one row of trees at a time (i.e. hit a key between rows). You may also resize the output window and/or use the scroll bar to show the entire output during judging. If you are unable to set your output window so that it uses a fixed width font, you may copy and paste your output into notepad or a similar text editor during judging.

Sample Input

```
Since the Greeks invented logic
and geometry the idea that all reasoning
might be reduced to some kind of
calculation has fascinated most of
the West's rigorous thinkers
```

Sample Output

See the previous page. Note that each new row of trees should appear in the output window *below* the row before, not beside it as shown here. The rows are only shown side by side to save space.

Problem 2: Jewelry Tips

There's a popular online game where the player gets an 8x8 board of coloured jewels and their goal is to form horizontal or vertical lines of 3 or more jewels of the same colour. They do this by repeatedly swapping pairs of adjacent jewels, either horizontally or vertically. A swap is only allowed if it would create at least 1 line of at least 3 jewels of the same colour. There are 7 colours: Red, Orange, Yellow, Green, Blue, Purple, and White.

Sometimes the players get stuck and have to be given a tip on what to do next. The tips fall into 3 categories: Normal, Good, and Excellent. A tip is "Normal" if acting on it would create a single line of 3 same-coloured jewels in a row. A tip is "Good" if acting on it would create a single line of 4 same-coloured jewels in a row. A tip is "Excellent" if acting on it would create either more than one line or a single line of 5. Here are some example boards. Underneath each board is the tip the game would give in that case.

```
BWPROYY
GBRBGWO
BPOOPPP
BWGYGWB
YGBYOBG
RGPOGOO
YBBYYOG
PBRYGPP
```

Norm: B.DN@0,0

```
YWPRRWGR
WGGPWOWP
PGWYOWO
YBWRGWP
WRGOYGP
GYPRWBB
BGWOPOB
ORRBBYP
```

Good: W.RT@1,4

```
GBWYPPOR
PPGWBYGP
OBORRWPR
RPGGOPOW
WBYYBBRP
GWPOWRGO
RPYOWPGO
RBWPGBOG
```

Excl: O.RT@7,6

Each tip starts with a four character coding of its type (`Norm`, `Good`, or `Excl`), then a colon and a space, then an 8 character code that gives the colour of the jewel to be swapped, the direction to swap it, and the location of the jewel before the swap. So "`Good: W.RT@1,4`" means "A good move is to take the white jewel at row 1, column 4 and swap it with the jewel to the right". The directions (up, down, left, and right) should be given using the two-character codes `UP`, `DN`, `LT`, and `RT`.

The colour of the jewel named in the tip must always match the colour of one of the lines that will be formed. For example, the tip in the first example could have been given as "`Norm: G.UP@1,0`", but the system would never express the tip this way because the line that would be formed if the player used the tip would be Blue, not Green.

Note that in the above examples, the first tip is "Normal" because it creates a single line of 3 blue jewels and no other lines, the second is "Good" because it will create a single line of 4 white jewels and no other lines, and the third is "Excellent" because it will create two lines of 3 jewels – one green and one orange.

The game's tipping system follows a few basic rules to decide which tip to show, out of all the possible tips it could show.

Rules for Tipping

1. Don't give a good or excellent tip if there is a normal tip available.
2. Don't give an excellent tip if there is a good tip available.
3. When choosing between two tips of the same type on different rows, always choose the one from the highest row.
4. When choosing between two tips of the same type on the same row, always choose the one from the leftmost column.
5. In a situation where there is more than one possible tip of the same type for the same jewel, give priority to LT tips, then UP, RT and DN in that order.

In the first example above, other possible tips include "`Good: O.DN@4,4`" and "`Norm: O.RT@5,3`", but the first is prohibited by rule 1 and the second is prohibited by rule 3. In the second example, another possible tip would be "`Excl: W.DN@0,5`", but this is prohibited by rule 2. And in the third example, another possible tip would be "`Excl: G.LT@7,7`" but this is prohibited by rule 4.

DATA21.txt (DATA22.txt for the second try) will contain 10 test cases. Each test case is an 8x8 game board as shown below (note that the sample input shown only has 5 test cases). None of these game boards will contain a row of 3 or more jewels of the same colour. Your task is to write a program that will output a tip for each board, according to the specifications outlined above. If there are no tips to give, the program should simply output "Game Over".

Sample Input

```
YYBORBWG  GWOGWOGO  BOBWOPRP  OWBGYOYG  GGRBYWBB
RBPYRORY  PWOPGWRY  YORPBGOW  GGOYBGPW  RYWWOYWY
WYGBYPBO  PYBPBRRW  WWGYWWGB  WBPGRGWW  YYOBOPOP
BBRPBGPW  BOWGBBOG  RBRBWBOW  OPPWWRYP  WGBPBBYP
PWPYROBP  GWROGPWG  YWWGGRWR  YBORGOOP  RWRBYWGW
PYYRYWWR  PYPWBRRY  OGWRGBGB  WWOOWWRG  BROBOYRB
GPBYRYYG  POGRGPOO  BBPPWWOP  PPYBYPOG  OWGWPROP
PPWGOGPB  RYYOGBPW  WWPBORPB  RGOYPOWW  RWGWPGOO
```

Sample Output

```
Norm: P.RT@5,0
Excl: G.DN@4,4
Game Over
Norm: O.UP@7,2
Good: B.LT@3,4
```

Problem 3: Steam Arithmetic

Steam is a brand new dialect of Lisp, one of the oldest high-level programming languages. The name Lisp comes from the phrase "List Processing" because the only structure in Lisp (and therefore in Steam) is the list. A list is a sequence of tokens separated by whitespace and enclosed in round brackets. Here is a list with 6 elements: `(6 z 4.2 hello 4 world)`.

Lists can be nested, meaning they can have other lists as elements. Here is an example of a nested list: `(5 (x t e d) (6 7 (-1 2)))`. This list has three elements. The first is 5, the second is the list `(x t e d)`, and the third is the list `(6 7 (-1 2))`, which also has another list nested inside of it. There is no limit to how many lists can be nested inside other lists.

Arithmetic in Steam uses "prefix" notation, in which the operator is written before the operands. We are used to seeing arithmetic in "infix" notation like this: 5+3. But in prefix notation `"5 + 3"` becomes `"+ 5 3"`. In Steam, arithmetic expressions are lists like this:

`(+ 5 3)`	means	`5 + 3`
`(* 7 2)`	means	`7 * 2`
`(* (- 5 2) (+ 2 3))`	means	`(5 - 2) * (2 + 3)`

In Steam arithmetic, there are 5 operators: +, -, *, q (for quotient) and r (for remainder, which is the same as the standard "modulus" or "mod" operator for integers). Each operator takes exactly two operands, and the only possible operands are the integers 0 through 9. Every element in a list except for the last one is followed by a single space, and no other spaces are allowed anywhere in the list.

Note: You are not permitted to solve this problem using any dialect of Lisp as your programming language.

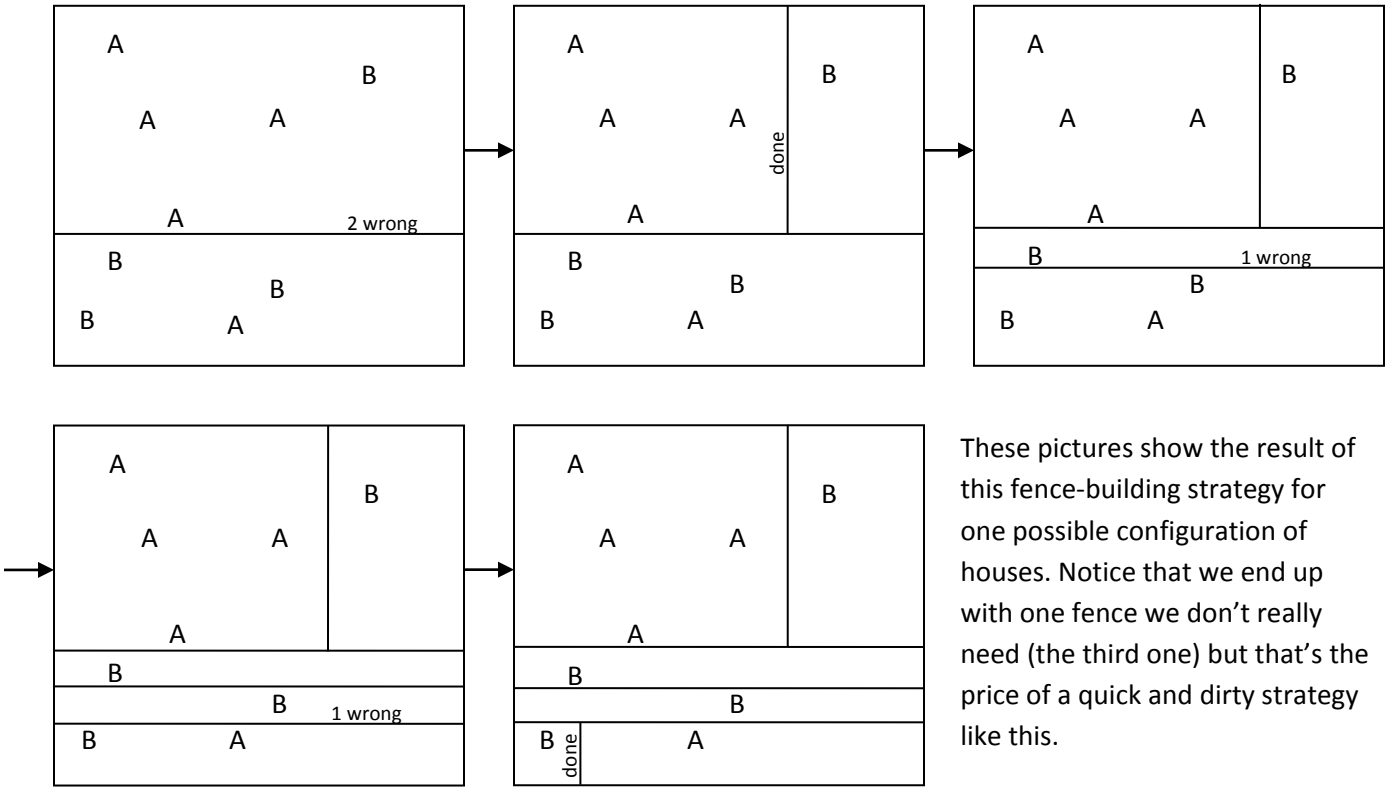
DATA31.txt (DATA32.txt for the second try) will contain 10 syntactically correct arithmetic expressions written in Steam. Your job is to evaluate each expression and output the result. The result could be any positive or negative integer in the range $-32{,}768$ to $32{,}767$.

Sample Input	Sample Output
`(+ 5 3)`	8
`(* 7 2)`	14
`(- 0 9)`	-9
`(q 8 5)`	1
`(r 8 5)`	3
`(* (- 5 2) (+ 2 3))`	15
`(r (q 9 4) (* (+ (q 2 3) (- 5 (- 4 (* 2 3)))) 2))`	2
`(* 4 (* 4 (* 4 (* 4 (* 4 (* 4 4))))))`	16384
`(* (* (* (* (* (* 4 4) 4) 4) 4) 4) 4)`	16384
`(+ (q (- 9 3) (r 7 4)) (- (* 9 9) (* 0 1)))`	83

Problem 4: Splitsville

A terrible feud has broken out in the French commune of Lacelle between families that use Android phones (A) and families that use Blackberries (B). Steps need to be taken immediately to keep the two groups apart. We have a map of where everyone lives and an unlimited budget to build electric fences. We don't have time to plan the best possible use of our fences, so we will go with a simple strategy that seems to get pretty good results most of the time.

1. We will only use fences running North to South (vertical) or East to West (horizontal).
2. When separating a region of the town, we will build the fence that best separates the houses in that region, creating two new regions. Then we will divide the new regions.
3. We will continue in this way until every region contains houses of only one family.
4. When deciding on the "best" fence in a region, we will go with the one that separates at least one house from the others and results in the fewest number of houses on the wrong side of the fence. (A house on the wrong side of the fence is a B house in a majority A area or an A house in a majority B area. If a region has the same number of A and B houses, then the A houses are considered to be on the wrong side.)
5. If there is a tie for the best fence between horizontal and vertical fences, we will use a horizontal one. If there is a tie among horizontal fences, we will use the Northernmost one. If there is a tie among vertical fences, we will use the Westernmost one.



These pictures show the result of this fence-building strategy for one possible configuration of houses. Notice that we end up with one fence we don't really need (the third one) but that's the price of a quick and dirty strategy like this.

DATA41.txt (DATA42.txt for the second try) will contain five test cases. Each test case consists of two lists of **X** and **Y** coordinates – one for the Android families, and one for the Blackberry families. Each list starts with the number of items in the list on a single line, followed by the **X** and **Y** locations of each house, one per line. The **X** coordinates are the East-West coordinates, with higher values being more Eastern locations. The **Y** coordinates are the North-South coordinates, with higher values being more Northern locations. Coordinates are integers ranging from -100 000 to +100 000.

Your output should consist of one number for each test case representing the number of fences you have to build in order to separate the families, following the strategy described above. The first test case below represents the example pictured above. There will always be at least one and at most 200 houses in the village. The maximum width and height of the village is 10000 units. No two houses will ever share the same coordinates.

Sample Input

```
5
-4 4
-3 2
1 2
-1 -1
0 -4
4
-5 -4
-4 -2
1 -3
3 3
6
-3 4
5 6
2 4
-6 -6
-5 3
0 0
6
```

```
0 1
12 4
6 6
-7 -8
3 -5
10 10
10
-1 3
-1 4
-1 6
-1 8
-1 -100
-1 -7
-1 -15
-1 -17
-1 9
-1 -5
2
1 1
```

```
1 -1
2
1 1
-1 -1
2
1 -1
-1 1
5
0 3
-2 5
-3 4
-5 6
-7 8
4
-3 5
1 2
2 3
3 4
```

Sample Output

```
5
8
1
3
5
```