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 its own own radar tower. 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

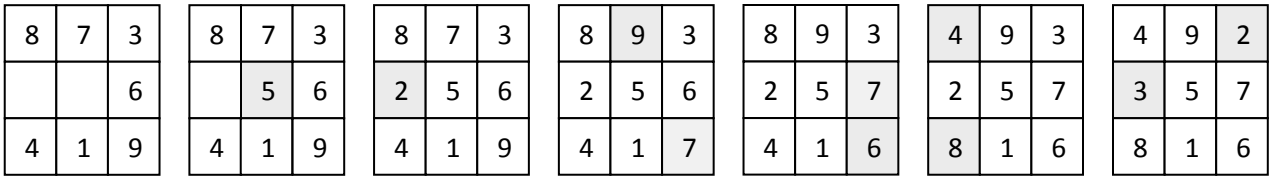
8	5	6
5	7	7
4	6	7

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
---
```