DKC^3 2018 – Programming Problems

1. Chinese Phone Numbers (25 points)

In China, phone numbers are 11 digits, like: 15012233444. The numbers can be grouped in many ways. Some group the numbers into 3-4-4 format, i.e. 150 1223 3444. While some group the numbers into 3-3-5 format, i.e. 150 122 33444. Other groupings also exist. Different groupings lead to different ways of reading these numbers aloud.

Given a list of phone numbers and the dividing formats, output the right ways to read these numbers.

Rules:

Single numbers, just read them separately.
2 successive numbers, use double.
3 successive numbers, use triple.
4 successive numbers, use quadruple.
5 successive numbers, use quintuple.
6 successive numbers, use sextuple.
7 successive numbers, use septuple.
8 successive numbers, use octuple.
9 successive numbers, use nonuple.
10 successive numbers, use decuple.
More than 10 successive numbers, read them all separately.

Input
There will be 10 test cases. Each line contains a phone number **N**, where 1 ≤ **length of N** ≤ 100 and the dividing format **F**, one or more positive integers separated by dashes (-), without leading zeros and whose sum always equals the number of digits in the phone number.

Output
For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is the reading sentence in English whose words are separated by a space.

Input File: C:\DKC3\PhoneIn.txt
Output File: C:\DKC3\PhoneOut.txt

Examples:

Input:
15012233444 3-4-4
15012233444 3-3-5

Output:
Case #1: one five zero one double two three three triple four
Case #2: one five zero one double two double three triple four

DKC^3 2018 – Programming Problems

2. Sudoku Checker (45 points)

Sudoku is a popular single player game. The objective is to fill a 9x9 matrix with digits so that each column, each row, and all 9 non-overlapping 3x3 sub-matrices contain all of the digits from 1 through 9. Each 9x9 matrix is partially completed at the start of game play and typically has a unique solution.

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

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

Given a completed 9x9 Sudoku matrix, your task is to determine whether it is a *valid* solution.
A *valid* solution must satisfy the following criteria:

- **Rule 1:** Each row contains each number from **1** to **9**, once each.
- **Rule 2:** Each column contains each number from **1** to **9**, once each.
- **Rule 3:** Divide the **9x9** matrix into **9** non-overlapping **3x3** sub-matrices. Each sub-matrix contains each number from **1** to **9**, once each.

If the completed matrix is not valid, you must determine which criteria (see above) were broken.

Input
Each test case starts with an integer **N** where N is the number of test case.

Output
For each test case, output one line containing "Case #x: y", where x is the case number (starting from 1) and y is "Yes" (quotes for clarity only) if it is a valid solution, or "No" (quotes for clarity only) if it is invalid, followed by the number of the specific criteria that were not met. In the case that multiple criteria were not met, the numbers of the criteria should appear in numerical order – lowest to highest. If the same criterion is not met several times, it should only appear once in the output. Note that the judge is case-sensitive, so answers of "yes" and "no" will not be accepted.

Input File: C:\DKC3\SudokuIn.txt
Output File: C:\DKC3\SudokuOut.txt

Examples:

Input:

```
1
534678912
672195348
198342567
859761423
426853791
713924856
961537284
287419635
345286179
2
354916728
689745136
712238954
972463815
148597263
635182479
567824391
893671542
421359687
```

Output:

```
Case #1: Yes
Case #2: No 1 2
```

3. Centauri Prime (5 points)

Back in the old days before the creation of the mighty Centauri Republic, the planet Centauri Prime was split into several independent kingdoms. The kingdom of Mollaristan was ruled by king Loatold, while the kingdom of Auritania was under the rule of queen Elana. In fact, it just so happened that every kingdom whose name ended in a consonant was ruled by a king, while every kingdom whose name ended in a vowel was ruled by a queen. Also because of an amazing coincidence, all kingdoms whose named ended in the letter 'y' were constantly in a state of turmoil and were not ruled by anyone. Can you write a program that will determine the current rulers of several countries, given the countries' names?

Input
There will be 10 test cases. Each line contains the name of one country. Country names will consist of only lower case English letters, starting with a capital letter. There will be no other characters on any line, and no empty lines. Each country name will have at most 100 characters.

Output
For each test case, output one line containing "Case #**x**: **C** is ruled by **Y.**", where **x** is the case number (starting from 1), **C** is the country name, and **Y** is either "a King", "a Queen" or "nobody". Be careful with capitalization and the terminating period. Your output must be in exactly this format.

Input File: C:\DKC3\CentauriIn.txt
Output File: C:\DKC3\CentauriOut.txt

Examples:

Input:
Mollaristan
Auritania

Output:
Case #1: Mollaristan is ruled by a King.
Case #2: Auritania is ruled by a Queen.

4. Name Scores (15 points)

Write a program that assigns a score to a name in regard to how it fits into a list of other names.

Input
You will be given a name *N* followed by a list of names. Each name will be on its own line. There will be an empty line between test cases.

Output
Begin by working out the alphabetical value of *N* by finding the value for each letter in *N* (for example: a = 1, b = 2, and so on). Then sort the list into alphabetical order and determine where *N* would fit alphabetically in the list. Next, multiply the alphabetical value of *N* with its order in the list to get the score. Output each *N* followed by a space, a hyphen, another space, and its score. Each test case answer should be on a separate line. Be sure to use commas, apostrophes, and other grammatical symbols where appropriate.

For example, when the list is sorted into alphabetical order, COLIN, which is worth 3 + 15 + 12 + 9 + 14 = 53, is the 9th name in the list. So, COLIN would obtain a score of 9 × 53 = 477.

Input File: C:\DKC3\NamesIn.txt
Output File: C:\DKC3\NamesOut.txt

Examples:

Input:
Colin
Marty
Byron
Edward
Marian
Avery
Aaron
Carl
Beverly
Adrian
Brian
Paul
Carry

Josh
Randall
Emily
Shelly
Peter
Ryan

Output:
Colin – 477
Josh – 104

5. Clock Hands (10 points)

Find the angle between the minute hand and the hour hand on a regular analog clock. Assume that the second hand, if there were one, would be pointing straight up at the 12. Give all angles as the smallest positive angles. For example, 9:00 is 90 degrees; not -90 or 270 degrees.

Input
Each test case consists of a time in the form '*H*:*M*', with $1 \le H \le 12$ and $00 \le M \le 59$. Note that *H* may be represented with 1 or 2 digits (for 1-9 or 10-12, respectively); *M* is always represented with 2 digits, just like an analog clock. Each test case is separated with a blank line.

Output
For each time in the test cases, you should output the smallest positive angle in degrees between the minute and hour hands of the clock. The answer should be between 0 degrees and 180 degrees for all input times. Display each angle on its own line in the same order as the input. The output should be rounded to the nearest 1/1000 (3 places after the decimal point should be printed).

Input File: C:\DKC3\ClockIn.txt
Output File: C:\DKC3\ClockOut.txt

Examples:

Input:
12:00
6:45

Output:
0.000
67.500

6. WERTYU (5 points)

A common typing error is to place the hands on the keyboard one row to the right of the home keys. So 'Q' is typed as 'W', 'J' is typed as 'K', and so on. Decode a message written in this manner.

Input
Each test case will consist of one line and represents a message to decode. Each message may contain digits, spaces, uppercase letters (except Q, A, Z), and the following punctuation: left/right brackets ([ ]), backslash (\), semi-colon (;), apostrophe (‘), comma (,), period (.), and forward slash (/). Keys labelled with words (Tab, Backspace, Enter, etc.) are not represented in the input.

Output
You are to replace each character with the one immediately to its left on the keyboard shown above with the given restrictions. Print out each decoded message on a new line. Spaces in the input should be echoed in the output.

Input File: C:\DKC3\WertyuIn.txt
Output File: C:\DKC3\WertyuOut.txt

Examples:

Input:
O S, GOMR YPFSU/
FLV4’ VP;;RHOSYR VP,[IYOMH VP,[RYOYOPM

Output:
I AM FINE TODAY.
DKC3; COLLEGIATE COMPUTING COMPETITION

7. Light on (25 points)

There is a light in a room which lights up only when someone is in the room (think motion detector). You are given a set of intervals in entrance and exit times as single integers, and expected to find how long the light has been on. When the times overlap, you need to find the time between the smallest and the biggest numbers in that interval.

Input
You'll be given a set of two integers per line telling you the time points that someone entered and exited the room, respectively. Each line is a visitor, each block is a room (a test case). There will be ten rooms. The end of each room in the input data will be delimited by a single dash (-) character.

Output
Your program should report the number of hours the lights would be on. Each room should print a single integer on its own line.

Input File: C:\DKC3\LightIn.txt
Output File: C:\DKC3\LightOut.txt

Examples:

Input:

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

Output:

```
3
7
```

DKC^3 2018 – Programming Problems

8. Packet Assembler (40 points)

When a message is transmitted over the internet, it is split into multiple packets. Each packet is transferred individually, and the packets are reassembled into the original message by the receiver. Because the internet exists in the real world, and because the real world can be messy, packets do not always arrive in the order in which they are sent. For today's challenge, your program must collect packets, assemble them in the correct order, and print the completed messages in the order which they are completed.

While you will be reading from a fixed file, the point of this exercise is to simulate incoming packets. For the purposes of this challenge, assume there is a potentially unlimited number of packets.

Input:
The incoming packet stream will be simulated via a fixed file. Each line of input represents a single packet. Each line will be formatted as X Y Z *some text*, where X Y and Z are positive integers and *some text* is an arbitrary string. X represents the message ID (i.e. which message this packet is a part of). Y represents the packet ID (i.e. the index of this packet in the message). Z represents the total number of packets in the message. Packets are zero-indexed, so the first packet in a message will have Y=0, the last packet in a message will have Y=Z-1.

For this challenge, there will be no “retransmissions” or lost packets - it is guaranteed that there will be no duplicate packets or message IDs, and that all packets will eventually arrive.

Output:
Output each completed message, one line per packet. Messages should be outputted in the order in which they are completed.

Input File: C:\DKC3\PacketIn.txt
Output File: C:\DKC3\PacketOut.txt

Examples:

Input:

```
6220    1    10    Because he's the hero Gotham deserves,
6220    9    10
5181    5    7     in time, like tears in rain. Time to die.
6220    3    10    So we'll hunt him.
6220    5    10    Because he's not a hero.
5181    6    7
5181    2    7     shoulder of Orion. I watched C-beams
5181    4    7     Gate. All those moments will be lost
6220    6    10    He's a silent guardian.
5181    3    7     glitter in the dark near the Tannhauser
6220    7    10    A watchful protector.
5181    1    7     believe. Attack ships on fire off the
6220    0    10    We have to chase him.
```

5181	0	7	I've seen things you people wouldn't
6220	4	10	Because he can take it.
6220	2	10	but not the one it needs right now.
6220	8	10	A Dark Knight.

Output:

5181	0	7	I've seen things you people wouldn't
5181	1	7	believe. Attack ships on fire off the
5181	2	7	shoulder of Orion. I watched C-beams
5181	3	7	glitter in the dark near the Tannhauser
5181	4	7	Gate. All those moments will be lost
5181	5	7	in time, like tears in rain. Time to die.
5181	6	7	
6220	0	10	We have to chase him.
6220	1	10	Because he's the hero Gotham deserves,
6220	2	10	but not the one it needs right now.
6220	3	10	So we'll hunt him.
6220	4	10	Because he can take it.
6220	5	10	Because he's not a hero.
6220	6	10	He's a silent guardian.
6220	7	10	A watchful protector.
6220	8	10	A Dark Knight.
6220	9	10	

9. Adding Reversed Numbers (15 points)

The Antique Comedians of Melanesia prefer comedies to tragedies. Unfortunately, most of the ancient plays are tragedies. Therefore, the dramatic advisor of ACM has decided to transfigure some tragedies into comedies. Obviously, this work is very hard because the basic sense of the play must be kept intact, although all the things change to their opposites. For example, the numbers: if any number appears in the tragedy, it must be converted to its reversed form before being accepted into the comedy play.

A reversed number is a number written in Arabic numerals but the order of digits is reversed. The first digit becomes last and vice versa. For example, if the main hero had 1245 strawberries in the tragedy, he has 5421 of them now. Note that all the leading zeros are omitted. That means if the number ends with a zero, the zero is lost by reversing (e.g. 1200 gives 21). Also note that the reversed number never has any trailing zeros.

ACM needs to perform calculations with reversed numbers. Your task is to add two reversed numbers and output their reversed sum. Of course, the result is not unique because any particular number is a reversed form of several numbers (e.g. 21 could be 12, 120 or 1200 before reversing).

Input
The input consists of 10 test cases. Each case consists of exactly one line with two positive integers separated by space.

Output
For each test case, print exactly one line containing only one integer, Omit any leading zeros in the output.

Input File: C:\DKC3\AddIn.txt
Output File: C:\DKC3\AddOut.txt

Examples:

Input:
24 1
4358 754

Output:
34
1998

DKC^3 2018 – Programming Problems

10. Phone Pad (10 points)

Write a program that will read in a list of phone number mnemonics (numbers represented as full or partial words so they are easier to remember) and output the number in numerals.

Input
There will be one test case per line representing a phone number mnemonic. Characters and integers may be present in the mnemonic. Letters may be in mixed case. Ignore dashes, periods and spaces in the input. There are 10 total test cases.

Output
Use the phone pad diagram above to determine which letters to associate with each number. If a numeric phone number cannot be generated from the mnemonic, output the word "INVALID."

Input File: C:\DKC3\PadIn .txt
Output File: C:\DKC3\PadOut .txt

Examples:

Input:
1.800.DIGI.KEY
BUY-MORE-NOW

Output:
18003444539
INVALID

11. Check Printing (20 points)

Professor Napolean Pudge the great modern French Mathematician was asked to be a guest professor at a prestigious University in the midwestern United States. While in America he decided to open a checking account with Goods Moorhead Bank, a large bank in the U.S. After opening the checking account he realized that while he loved writing numbers he found the process of writing out the dollar amount out on his checks very tedious. To help him out, write a program that takes a dollar amount in 2 digit decimal format and prints out the dollar amount in words, including the dollar and cents labels.

Input
Each test case will represent a check amount in digits. There are 10 total test cases.

Output
Print out each dollar amount in words. If the value is greater than the amount he has in his checking account, which is always $5000.00, print OVERDRAFT WARNING instead of the dollar amount.

Input File: C:\DKC3\CheckIn.txt
Output File: C:\DKC3\CheckOut.txt

Examples:

Input:
3443.16
3299999.21

Output:
three thousand four hundred forty three dollars and sixteen cents
OVERDRAFT WARNING

12. Shortest Substring (30 points)

Given a string made from an alphabet α of characters, you must find the shortest substring of that string that contains at least one of each character in α. If there are more than one such substring tied for shortest, the one that occurs earliest will be the answer. There will be ten test cases.

Input
Each test case will be a line containing a string of characters corresponding to the alphabet α for that test case followed by a string to evaluate. The string to be evaluated will never contain characters that are not in the alphabet and will always contain at least one of every character in the alphabet. Furthermore, the alphabets will only contain lowercase letters, uppercase letters and numbers.

Output
The output for each test case will be the case number and the shortest substring containing every character of the alphabet.

Input File: C:\DKC3\ShortIn.txt
Output File: C:\DKC3\ShortOut.txt

Examples:

Input:
abcdefghijklmnopqrstuvwxyz
thequickbrownfoxjumpsoverthelazydog
123456789
3141592653589793238462643

Output:
Case 1: quickbrownfoxjumpsoverthelazydog
Case 2: 415926535897

13. Knight Moves (45 points)

Given a lattice point in the Euclidean plane, find the fewest number of chess knight moves (up 1 over 2, etc) required to reach that point from the origin. There will be ten test cases.

Input
Each test case will consist of two numbers giving the x and y coordinates, respectively, of a lattice point. Neither coordinate will be greater than one billion in magnitude.

Output
The output should be a case number followed by the minimum required chess knight moves to reach that point from the origin.

Input File: C:\DKC3\KnightIn.txt
Output File: C:\DKC3\KnightOut.txt

Examples:

Input:
2 3
10 0

Output:
Case 1: 3
Case 2: 6

14. Circumcircle (35 points)

Given a set of three points, determine the equation of the circle that passes through those points, if such a circle exists. Hint: the center of a triangle's circumcircle is at the intersection of the perpendicular bisectors of the triangle's sides. There will be exactly ten test cases.

Input
Each test case will be on its own line and will consist of three coordinate pairs in (x,y) format, with a space separating each pair.

Output
The output for each test case should be the case number followed by the equation of a circle in standard form ($(x-h)^2+(y-k)^2=r^2$ where (h,k) is the center of the circle and r is the radius). All numbers should be rounded and expressed to the nearest hundredth. If it is impossible for a circle to pass through all three points, print IMPOSSIBLE.

Input File: C:\DKC3\CircleIn.txt
Output File: C:\DKC3\CircleIn.txt

Examples:

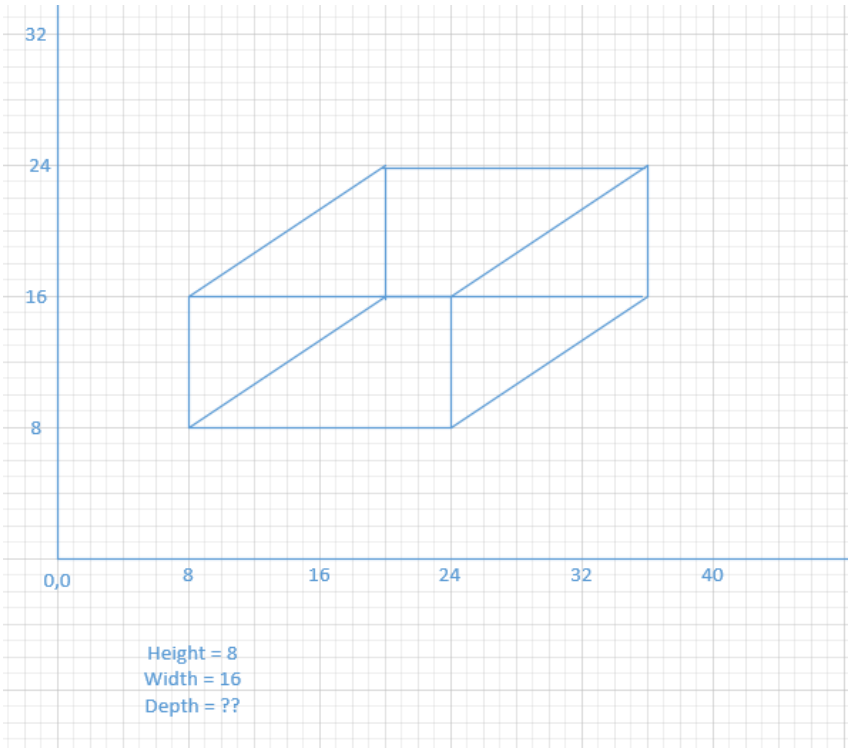
Input:
(0,0) (2,1) (3,4)
(0,0) (1,1) (5,5)

Output:
Case 1: (x + 0.50)^2 + (y - 3.50)^2 = 12.50
Case 2: IMPOSSIBLE

DKC^3 2018 – Programming Problems

15. Box Volume (20 points)

You will be given the 8 2D coordinates for the corners of a 3D box. Using those coordinates calculate the volume of the given box. Example picture shown below.



Input
You will be given 10 lines of data. One for each test case. Each line will consist of the 8 corner coordinates of the 3D box. The coordinates will always be integer values. The "depth" of the box will be the distance between two corresponding corners on the front and back of the box. In the example the "depth" is the distance between (8,8) and (20,16).

Output
Round the volume of the box to the nearest whole number.

Input Files: C:\DKC3\BoxIn.txt
Output Files: C:\DKC3\BoxOut.txt

Examples:

Input:
(8,8) (8,16) (24,8) (24,16) (20,16) (20,24) (36,16) (36,24)
(0,0) (0,6) (6,0) (6,6) (3,3) (9,3) (3,9) (9,9)

Output:
1846
153