DKC^3 2022 - Short Programming Problems

Important Reminders:

- Time remaining will be announced at 30 min., 5 min. and 1 min. left in the session.

- Assume leading zeros are used in all decimal notations unless the problem says otherwise. (ex: 0.167)

- Example input and output files are available in the **C:\DKC3** folder for each problem. These may be used to test your programs.

- Each program must read from its respective input file in the **C:\DKC3** folder and generate an output file in that same folder. The names of these files must match what is specified for each problem in this document. These output files will be retrieved electronically and scored by the judges at the end of the session.

- Do **NOT** store any code in the **C:\DKC3** folder. Anything stored there will be overwritten.

- At the end of the session, judges will overwrite the example input files with the official input files for each problem. Each file will contain 10 test cases. One member of each team will be asked by a competition organizer to run all completed programs. Programs may be recompiled at this point prior to being run, but no changes can be made to source code.

- Each program must complete execution within **two** minutes.

- After running your programs, be sure to close out of all IDEs.

DKC³ 2022 - Short Programming Problems

1. Prime Path (30 points)

You are designing a robot that reads the value of floor tiles before deciding how to move. While robots that only move to even or only move to odd numbers were easy, the prime number bot has been problematic. We need it to move along paths up, down, left, or right. The robot starts in the upper left corner and attempts to move to the goal - the lower right corner. The complication: It should only move to tiles with a prime number.

Input
Input will be a grid of numbers, 10 x 10. Numbers are one to four digits, separated by a space, ten numbers to a line, 10 lines per problem. There will be 10 test cases, each separated by an asterisk. (Note: The numbers in the examples below are aligned, and the primes are in bold, and the shortest path is underlined. This is for visual example only.)

Output
Output will be the list of moves to get from start to finish, each separated by a space, using the following codes: R – Move to the right L – Move to the left U – Move up D – Move down

Input File: C:\DKC3\PrimeIn.txt
Output File: C:\DKC3\PrimeOut.txt

Examples:
Input:

2	**3**	4	181	6	**823**	1166	38	**1709**	48
8	**5**	**7**	**19**	10	87	85	8446	843	45
22	24	18	**29**	24	30	118	**1601**	2862	1815
44	55	28	**71**	62	45	12	453	7456	846
10	20	58	**31**	**61**	**79**	8	183	**223**	428
78	**4423**	87	1035	7745	**97**	**83**	2288	413	833
118	4538	896	16	32	81	**59**	638	996	1184
459	123	841	713	8	**811**	**67**	742	4632	845
13	784	663	842	8	**1051**	1037	**43**	1336	4568
136	99	**463**	6	458	**37**	**823**	**89**	**709**	**719**
*									
2	**3**	4	181	**67**	**823**	**1669**	**389**	**1709**	48
8	**5**	**7**	15	**107**	87	85	**7457**	843	45
22	**17**	18	**29**	**23**	30	118	**1601**	2862	1815
44	**157**	28	72	**167**	45	12	**457**	7456	846
10	**19**	**5**	**31**	**61**	178	8	**181**	**223**	428
78	**4423**	87	1035	7745	**97**	**83**	**2287**	413	833
118	4538	896	16	32	81	**59**	638	996	1184
459	123	841	713	8	**811**	**67**	742	4632	845
13	784	663	842	8	1052	**1039**	435	1336	4568
136	99	**463**	6	458	**37**	**823**	**89**	**709**	**719**
*									

Output:

R D R R D D D R R D R D D L D D R R R R
R D D D D R R R U U U U R R R D D D D D L D D D D R R R

2. Number Triangle **(20 points)**

Write a program that calculates the highest sum of numbers passed on a route that starts at the top and ends somewhere on the base. You may only move downwards to one of the two numbers diagonally below.

- The number of rows in the triangle is > 1 but < =100.
- The numbers in the triangle, all integers, are between 0 and 99.

The route in this case is shown in bold: 7, 3, 8, 7 and 5. The sum is 30.

7
3 8
8 1 0
2 **7** 4 4
4 **5** 2 6 5

Input
The input will consist of N, the number of rows, followed by N rows of digits that make up the triangle. The output is the highest sum that is reached. There will be 10 test cases, each separated by an asterisk.

Output
Print the highest sum of numbers.

Input File: **C:\DKC3\TriangleIn.txt**
Output File: **C:\DKC3\TriangleOut.txt**

Examples:
Input:
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
*
4
1
1 2
1 1 3
5 3 0 1
*

Output:
30
8

3. Look & Say (15 points)

The Look & Say sequence is an infinite sequence starting with the first term "1". Subsequent terms are generated by looking at the preluding term and saying what the term consists of. In other words, the nth term is generated by saying the (n-1)th term. As such, the second term in the sequence is "11", generated by saying the first term as "One 1" because there is one '1'. Continuing, the third term in the sequence is "21", generated by saying the second term as "Two 1" because there are two '1's.

Input
There will be 10 test cases, one per line. Each test case will contain an integer n greater than 0.

Output
For each test case, use the given input n to output the nth term in the Look & Say sequence.

Input File: C:\DKC3\LookSayIn.txt
Output File: C:\DKC3\LookSayOut.txt

Examples:
Input:
3
8

Output:
21
1113213211

DKC³ 2022 - Short Programming Problems

4. Knobs (5 points)

Digi-Key sells thousands of types of Knobs. The prices for six of Digi-Key's available knobs are as follows:

```
450-1719-ND              $1.97
226-4102-ND              $11.09
145-KSN6----N2--5---ND   $0.23
2419-L02-1537-ND         $6.59
RPC9346-ND               $0.85
2419-D905-ND             $3.35
```

Given a list of knob part numbers and quantities, calculate the total cost.

Input
The input will consist of 10, asterisk separated test cases. Each test case is a list of part numbers and their quantities. Each line in the list contains a single part number and quantity separated by a space.

Output
Output the labeled ($) total cost of the parts rounded to two decimal places in dollars.

Input File: C:\DKC3\KnobsIn.txt
Output File: C:\DKC3\KnobsOut.txt

Examples:
Input:
145-KSN6----N2--5---ND 6
226-4102-ND 2
450-1719-ND 5
*
145-KSN6----N2--5---ND 1
RPC9346-ND 11
2419-L02-1537-ND 7
2419-D905-ND 6

Output:
$33.41
$75.81

5. Range Extraction (10 points)

A format for expressing an ordered list of integers is to use a comma separated list of either individual integers or a range of integers denoted by the starting integer separated from the end integer in the range by a dash, '-'. The range includes all integers in the interval including both endpoints. It is not considered a range unless it spans at least 3 numbers. For example: 12,13,15-17.

Given a list of integers in increasing order, output the list and use ranges where appropriate.

Input
The input will consist of a comma separated list of increasing integers.

Output
Output the equivalent, comma separate list of integers and ranges.

Input File: C:\DKC3\RangeExtractionIn.txt
Output File: C:\DKC3\RangeExtractionOut.txt

Example:
Input:
-10,-9,-8,-6,-3,-2,-1,0,1,3,4,5,7,8,9,10,11,14,15,17,18,19,20
23,45,46,47,48,49,50,52,63

Output:
-10--8,-6,-3-1,3-5,7-11,14,15,17-20
23,45-50,52,63

6. Assembly Language (10 points)

Write a program that will execute a version of assembly language. This version of assembly language is composed of the following commands:

`store x y`	Store value x in memory location y
`add x y z`	Add the value in memory location x to the value in memory location y and place the result in memory location z
`sub x y z`	Subtract the value in memory location y from the value memory location x and place the result in memory location z
`mult x y z`	Multiply the value in memory location x with the value in memory location y and place the result in memory location z
`div x y z`	Divides the value in memory location x by the value in memory location y and place the result in memory location z – truncate any non-integer result to the next smallest integer.

Input
Input will consist of 10, asterisk separated test cases. Each test case contains one or more, one line commands. Assume each command follows valid syntax.

Output
Output the resulting integer at the memory location of the final instruction.

Input File: C:\DKC3\AssemblyIn.txt
Output File: C:\DKC3\AssemblyOut.txt

Examples:
Input:
```
store 2 0
store 3 1
mult 0 1 2
*
store -1 5
store 3 3
add 5 3 0
sub 0 3 2
div 0 2 4
```

Output:
```
6
-2
```

7. Absolute Difference (15 points)

You are given an array of X integers. Your goal is to remove elements from the array until the absolute difference between any integer and the element immediately before it is the same across the entire array.

Input
The input will consist of a comma separated list of integers.

Output
Output the minimum number of elements that need to be removed and the absolute value of the difference between each element and its predecessor. Separate the two values with a comma.

Input File: C:\DKC3\AbsoluteDifferenceIn.txt
Output File: C:\DKC3\AbsoluteDifferenceOut.txt

Examples:
Input:
5,8,9,11,15,14,17
6,9,7,6,6,5,6

Output:
2,3
3,0

8. Valued Strings (10 points)

Austin T. has come up with a novel way of assigning value to any given string of letters. For starters, each letter is worth the same as its position in the alphabet (A = 1, B = 2, C = 3, … , Z = 26). However, if two of the same letter appear consecutively, the value of each is multiplied by 2. If three appear consecutively their value is multiplied by 3, and so forth. For example, "AA" is worth 4 because there are two As so each is worth 2. The value of "CCCC" would be 48 because each C is worth 12 (its original value of 3 multiplied by 4 because it appears in a sequence of 4). Adding up the values of all the letters in a string yields the overall value of the string.

Input
Each input will consist of a single string of alphabetic characters, on its own line. These strings may be up to 1000 characters in length.

Output
The output should consist of a single integer, on its own line, denoting the value of the string.

Input File: C:\DKC3\ValuedStringsIn.txt
Output File: C:\DKC3\ValuedStringsOut.txt

Examples:
Input:
AARDVARK
DKCCCCCC

Output:
78
159

9. Sumstring (15 points)

Given an integer represented as a string, find the sum of all possible integer substrings.

Input
There will be 10 test cases, one per line. Each test case will contain an integer represented as a string.

Output
For each test case, output the sum of all possible integer substrings of the given integer.

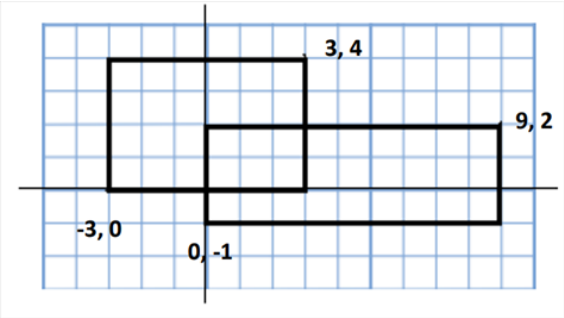
Input File: C:\DKC3\SumstringIn.txt
Output File: C:\DKC3\SumstringOut.txt

Examples:
Input:
1159
-320

Output:
1534
22

10. Overlapping Rectangles (15 points)

Given two rectangles that overlap, determine the total area of the two rectangles and the shared area of the two rectangles. A rectangle can be defined by the x and y coordinates of its lower left corner and its upper right corner. In this problem you will be given the coordinates to define two rectangles. The rectangles will intersect (overlap) in some way. For example, given the rectangles defined by rectangle 1 (-3, -0) and (3, 4) and rectangle 2 (0, -1) and (9, 2):



The area of a rectangle equals its width times its length. The area of the rectangle on the left is 24 (6 times 4). The area of the rectangle on the right is 27 (9 times 3). The area of the overlap section is 6. This means the total area of the rectangles is 45 (24 + 27 – 6).

Input
Each test case will consist of 2 lines of input, each representing the lower left and upper right corners of a rectangle. These coordinates will be in the format (x,y) and will be separated by a space. There are 10 test cases, with each test case separated by an asterisk (*).

Output
For each test case, output the area of the overlapping section and the total area the rectangles occupy. These values should be separated by a space.

Input File: **C:\DKC3\OverlapIn.txt**
Output File: **C:\DKC3\OverlapOut.txt**

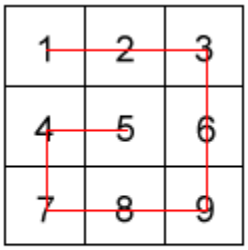
Examples:
Input:
(-3,0) (3,4)
(0,-1) (9,2)
*
(0,0) (3,3)
(2,2) (4,4)

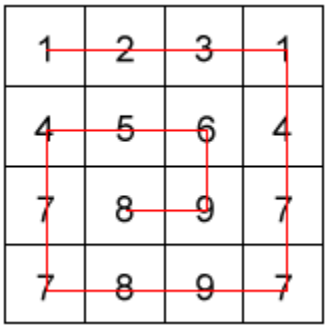
Output:
6 45
1 12

DKC³ 2022 - Short Programming Problems

11. Snail Shell Sort (5 points)

Given an N x N grid of natural numbers, output the elements arranged from outermost elements to the middle element, traveling clockwise.





Input
The input will consist of comma separated natural numbers split onto multiple lines. Each line will represent a row in the grid. There will be an asterisk on a newline separating the grid for each test case.

Output
Output a comma separated list of grid elements arranged from outermost elements to the middle element, traveling clockwise.

Input File: **C:\DKC3\SnailShellSortIn.txt**
Output File: **C:\DKC3\SnailShellSortOut.txt**

Example:
Input:
1,2,3
4,5,6
7,8,9
*
1,2,3,1
4,5,6,4
7,8,9,7
7,8,9,7

Output:
1,2,3,6,9,8,7,4,5
1,2,3,1,4,7,7,9,8,7,7,4,5,6,9,8

12. Largest Triangle (20 points)

Given a list of lattice points in (x, y) format, find the largest triangle by area that is formed by any three points in the list.

Input
Each test case will consist of a single line of (x, y) coordinates, each separated by a space. There will be at least 3 non-colinear points per test case, and there may be up to 100 total points per test case. Each point will be unique.

Output
The output for each test case should be the area, rounded and expressed to 2 decimal places, of the largest triangle that can be formed from 3 of the points in the list.

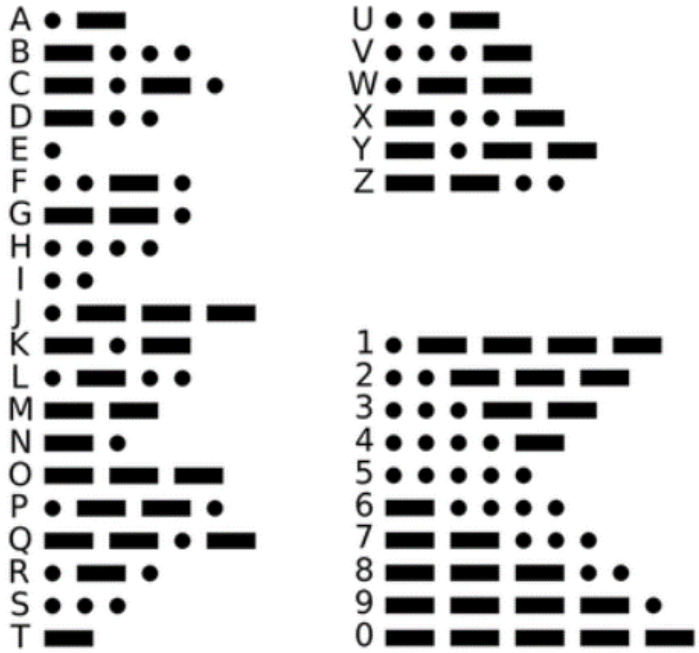
Input File: **C:\DKC3\LargestTriangleIn.txt**
Output File: **C:\DKC3\LargestTriangleOut.txt**

Examples:
Input:
(3, 10) (10, 12) (7, 7) (7, 10)
(1, 0) (-1, 9) (-9, -6) (9, 9)

Output:
14.50
75.00

13. Morse Code (10 points)

Write a program that converts Morse code to the English equivalent. Spaces indicate a new letter and two spaces indicate a new word. The outputs will not be case sensitive.



Input
There are 10 test cases. Each test case will be on a new line and can have up to 10 words.

Output
The output for each test case should be on separate lines and should be the word or phrase that the morse code input represents

Input File: **C:\DKC3\MoreCodeIn.txt**
Output File: **C:\DKC3\MorseCodeOut.txt**

Examples:
Input:
... --- ...
-.. .. --. .. -.- . -.--

Output:
sos
digi key

14. Reverse Directions (5 points)

A set of directions consists of several instructions. The first instruction is of the form "Begin on XXX", indicating the street that the route begins on. Each subsequent instruction is of the form "Left on XXX" or "Right on XXX", indicating a turn onto the specified road. When reversing directions, all left turns become right turns and vice versa, and the order of roads and turns is reversed. Reverse all directions for the given input leaving an empty line between each set of directions.

Input
There are 10 test cases. Each test case consists of 1 to 10 lines of input and each test case is separated by an asterisk.

Output
List of instructions to return to the original street. Separate each test case with a blank line.

Input File: C:\DKC3\ReverseIn.txt
Output File: C:\DKC3\ReverseOut.txt

Examples:
Input:
BEGIN on 28th St NW
LEFT on Hannah Ave NW
RIGHT on Paul Bunyan Dr NW
LEFT on HWY 2
RIGHT on HWY 371
*
BEGIN on 200th St
RIGHT on HWY 64
RIGHT on Broadway St W
LEFT on HWY 64
LEFT on HWY 200
RIGHT on HWY 71
LEFT on HWY 2
RIGHT on Paul Bunyan Dr NW
LEFT on Hannah Ave NW
RIGHT on 28th St NW

Output:
BEGIN on HWY 371
LEFT on HWY 2
RIGHT on Paul Bunyan Dr NW

LEFT on Hannah Ave NW
RIGHT on 28th St NW

BEGIN on 28th St NW
LEFT on Hannah Ave NW
RIGHT on Paul Bunyan Dr NW
LEFT on HWY 2
RIGHT on HWY 71
LEFT on HWY 200
RIGHT on HWY 64
RIGHT on Broadway St W
LEFT on HWY 64
LEFT on 200th St

15. Leaf Counting (25 points)

Austin E. has a program that reads in a list of numbers and stores them in a binary tree. For the purposes of this problem, consider a binary tree to be one where the left subtree of the root contains data that is less than or equal to the value of the data in the root, and the right subtree contains data that is greater than the value of the data in the root. The same will then hold for any node on the tree.

Austin's program adds numbers to the tree in the order they are received and does not perform any reordering or optimization. Given an input list of integers, you must determine how many leaves will be contained in the resulting tree (a leaf is a node with no sub-trees to either its right or left).

Input File: **C:\DKC3\LeafCountingIn.txt**
Output File: **C:\DKC3\LeafCountingOut.txt**

Examples:
Input:
0 5 9 -8 1 -1
2 2 2 2 2 2 2 3

Output:
3
2