DKC^3 2020 – 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\Short Programming** 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\Short Programming** 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.

- Please store your source code in the **C:\DKC3** folder. Do **NOT** store any code in the **C:\DKC3\Short Programming** subfolder. 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.

DKC^3 2020 – Short Programming Problems

1. Morse Code (15 points)

Morse code is a method used in communication to encode text characters as standardized sequences of two different signal durations, called dots and dashes. Morse code is named after Samuel Morse, an inventor of the telegraph.

The International Morse Code encodes the 26 English letters A through Z, some non-English letters, the Arabic numerals and a small set of punctuation and procedural signals. There is no distinction between upper and lowercase letters. Each Morse code symbol is formed by a sequence of dots and dashes. The dot duration is the basic unit of time measurement in Morse code transmission. The duration of a dash is three times the duration of a dot. Each dot or dash within a character is followed by period of signal absence, called a space, equal to the dot duration. The letters of a word are separated by a space of duration equal to three dots, and the words are separated by a space equal to seven dots.

To increase the efficiency of encoding, Morse code was designed so that the length of each symbol is approximately inverse to the frequency of occurrence of the character that it represents in text of the English language, thus the most common letter in English, the letter "E", has the shortest code: a single dot. Because the Morse code elements are specified by proportion rather than specific time durations, the code is usually transmitted at the highest rate that the receiver is capable of decoding.

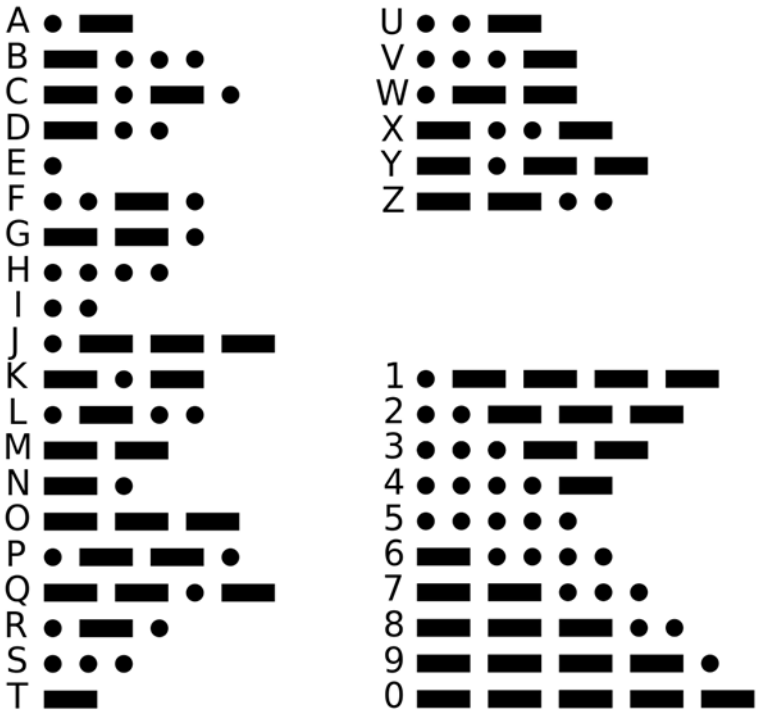
Input/Output information on the next page.

DKC³ 2020 – Short Programming Problems

Input

Each test case will be a line of Morse Code that you will need to translate into English. There will be 10 test cases. International Morse Code follows the following guidelines:

1. The length of a dot is 1 unit.	*For this problem this means 1 dot.*
2. The length of a dash is 3 units.	*For this problem this means 1 dash.*
3. The space between parts of the same letter is 1 unit.	*This means 1 space.*
4. The space between letters is 3 units.	*This means 3 spaces.*
5. The space between words is 7 units.	*This means 7 spaces.*



Output

There should be one line of output for each test case giving the English translation.

Input File: C:\DKC3\Short Programming\MorseIn.txt
Output File: C:\DKC3\Short Programming\MorseOut.txt

Examples:

Input:

```
... --- ...
-... .   ...- . .-. -.--   --.- ..- .. . -
```

Output:

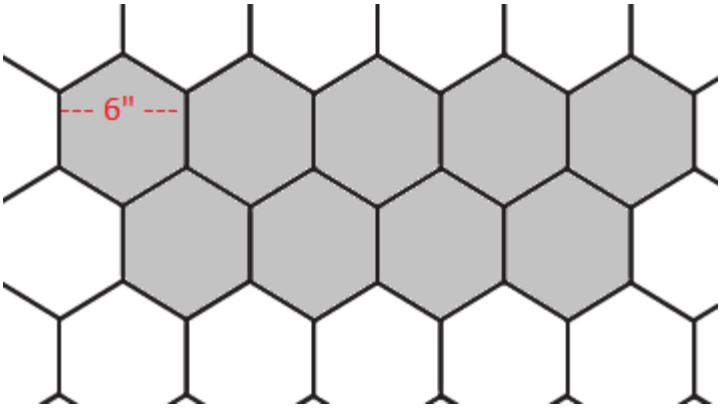
SOS
BE VERY QUIET

DKC^3 2020 – Short Programming Problems

2. Paving Stones (35 points)

A shop sells 6" hexagonal paving stones (image below). Each stone should be cemented into the ground with a 1/4" grout line between stones.

Customers call to place orders based on a rectangular area in feet. But some customers wish to completely fill the area while others would rather have a minimal gap to fill in around the edges.



Given an area and customer's edge preference, calculate how many paving stones to sell the customer at a minimum to cover their requirements without cutting any pavers.

Hexagon formulas where 'x' is our 6" side-to-side length:

- Apothem: a = x / 2
- Side: s = 2a / √3
- Angle-to-Angle: y = 4a / √3
- Perimeter: P = 6s
- Area: A = Pa / 2

Input/Output information on the next page.

Input
The input will consist of ten test cases, one per line. Each test case will specify the customer's area in the form "AxB" where A and B are either positive integers or positive decimals. Following the area and a space, the customer's edge preference will be specified with the text "fill" or "gap".

Output
The output for each test case should be a single line containing a positive whole number representing the minimum number of paving stones needed to cover the customer's area.

Input File: C:\DKC3\Short Programming\PavingStonesIn.txt
Output File: C:\DKC3\Short Programming\PavingStonesOut.txt

Examples:

Input:
0.5x0.5 fill
2x2 gap

Output:
3
12

DKC^3 2020 – Short Programming Problems

3. Abundancy Index (15 points)

In number theory, the abundancy index of a number is equal to the sum of the number's divisors divided by the number itself. Given a positive integer N, find the abundancy index of N rounded and expressed to 3 decimal places.

Input
Each test case will be on its own line and consist of a single positive integer N. There will be ten test cases.

Output
Each output should be the abundancy index of the number N, rounded and expressed to 3 decimal places.

Input File: C:\DKC3\Short Programming\AbundancyIn.txt
Output File: C:\DKC3\Short Programming\AbundancyOut.txt

Examples:

Input:
10
20

Output:
1.800
2.100

4. George's Grandiose Grammar (25 points)

George is known for creating odd grammatical orders for strings. Today he would like you to write a program that sorts a string into alphabetical order according to the following rules:

- Only the letters from A-Z are affected.
- The sorted output reflects the format of the original string with all non-alphabetic characters in their original positions.

Input
The input will consist of ten test cases, each on its own line. Each test case will contain a string with both alphabetic and non-alphabetic characters.

Output
The output consists of the original string in the input with the sorted string directly below it on a new line.

Input File: D:\DKC3\Short Programming\GeorgeGrammarIn.txt
Output File: D:\DKC3\Short Programming\GeorgeGrammarOut.txt

Examples:

Input:
THE PRICE OF BREAD IS $1.25 PER POUND.
THE LICENCE PLATE READ G76-ZA3.

Output:
THE PRICE OF BREAD IS $1.25 PER POUND.
ABC DDEEE EF HIINO OP $1.25 PPR RRSTU.
THE LICENCE PLATE READ G76-ZA3.
AAA CCDEEEE EGHIL LNPR T76-TZ3.

5. Simply Irreducible (20 points)

Write a program that takes in some number N and finds the sum of all unique, irreducible fractions that are less than N and contain only one or two-digit numbers in the numerator and denominator. Round your output to three decimal places.

Input
Input will consist of a single positive integer. There will be ten test cases, each one on its own line. Each test case will be separated by an asterisk (*).

Output
Output will consist of a decimal rounded to three places. Each line of output should be separated by an asterisk (*).

Input File: C:\DKC3\Short Programming\SimplyIrreducibleIn.txt
Output File: C:\DKC3\Short Programming\SimplyIrreducibleOut.txt

Examples:

Input:

1
*
2
*

Output:

1501.500
*
3582.951
*

DKC^3 2020 – Short Programming Problems

6. Hop To It (10 points)

While exploring a temple, Dr. Jones steps through a corridor to find a vast chasm with a Golden Idol on a platform near its center. To reach the Idol, the only path is a series of unstable stone columns protruding from the depths below that span from one end of the chasm to the other, but not directly through the central platform. A ladder will be required to cross from the closest column to the platform, and only certain columns can support any weight for more than a moment before crumbling. Trying to place the ladder on an unstable column will surely cause it to crumble and Dr. Jones to fall to his death.

Dr. Jones tests the first and second column and finds them to be stable, but not the third. Recalling some glyphs found along his journey, Dr. Jones believes the next stable column is always located **X** number of columns away from his current location, and **X** is also his current location from the ledge where he began.

Given a number (**N**) of columns between one end of the chasm to the other with the platform holding the Golden Idol being at the half way point, and a ladder long enough to reach from a column next or adjacent to the platform such that it would reach from column 4 or 5 if the platform were between the two; can Dr. Jones safely retrieve the Idol or will he fall to his doom?

Input
The input will consist of ten test cases, one per line. Each test case will be a single positive integer representing **N** number of columns**. 1 <= N <= 10000**.

Output
The output for each test case should be a single line with a **true** or **false** to indicate if he survived, followed by the number of **H** hops it took him to reach that outcome. These two values should be separated by a hyphen with spacing between.

Input File: C:\DKC3\Short Programming\HopToItIn.txt
Output File: C:\DKC3\Short Programming\HopToItOut.txt

Examples:

Input:
8
10

Output:
true - 4
false - 6

7. Combinational Logic (20 points)

Construct a 6-input gate which performs the following logical operation:

(not(A).B) + C.D + E.(not(F))

Where A, B, C, D, E, and F are the inputs to the 6-input gate, each period (".") resolves to a logical AND operation, while each plus ("+") resolves to a logical OR operation.

1 1 0 0 1 1 => (not(1).1) + 0.0 + 1.(not(1)) => 0.1 + 0.0 + 1.0 => 0 + 0 + 0 = 0
11111111 => (not(1).1) + 1.1 + 1.(not(1)) => 0.1 + 1.1 + 1.0 => 0 + 1 + 0 = 1

Input
The input will consist of ten test cases, one per line. Each test case will consist of 6 integers separated by single space. Each integer can be either a 1 or 0, so **0 <= A, B, C, D, E, F <= 1**.

Output
The output for each test case should be a single line with the final outcome of the 6-input gate followed by the decimal summation prior to applying the logical AND/OR operations with integers printed as whole numbers and decimals with the tenths. These two values should be separated by a hyphen with spacing between.

Input File: C:\DKC3\Short Programming\ComboLogicIn.txt
Output File: C:\DKC3\Short Programming\ComboLogicOut.txt

Examples:

Input:
1 1 0 0 1 1
1 1 1 1 1 1

Output:
0 - 1.1
1 - 2.2

8. Vampire Travel (45 points)

Nandor the Relentless loves to travel, but being a vampire has its obstacles. First, he can only travel by train because he must take his coffin with him. Second, he can only travel from dusk until dawn, namely 19:00 to 7:00. During the day, Nandor must stay inside a train station for obvious reasons. Because of this he usually brings along his vampire friends, Laszlo and Nadja, to keep him company. Third, he must take something to eat with him on his journeys. He needs one quart of blood daily, which he drinks at noon (12:00) inside his coffin. Help Nandor plan out his trips by finding the shortest route between two cities so that he can travel with the minimum amount of blood.

Nandor doesn't start planning his trips until after his daily meal at 12:00, so keep that in mind when scheduling his first departure and calculating the amount of blood he needs to bring with him. You can also assume that Nandor is able to fly and catch up with any train that departs at the same time he would be arriving on another train.

If more than one route will get Nandor to his destination at the same time, then he would prefer the route that has him on the trains for the least number of hours. If more than one route has the same destination time as well as the same number of hours on the trains, Nandor would prefer the route with the least number of transfers between trains.

Input
Each test case will consist of several route specifications. Each route specification contains the names of two cities, the departure time from the first city, and the total travel time in hours. Departure times and total travel times will always be whole numbers, and no route takes less than one hour or more than 24 hours. Keep in mind that Nandor cannot use routes departing earlier than 19:00 or arriving later than 7:00. The last line of each test case consists of two city names along with a travel time of zero (0). On this line, the first city is Nandor's start city and the second city is his destination. Each test case will be separated by an asterisk (*).

Output
For each test case you should output the order of travel Nandor should take, followed by the total number of hours he will be spending on the train(s). On a new line, output how many quarts of blood Nandor needs to take with him on his trip.

Input File: C:\DKC3\Short Programming\VampireIn.txt
Output File: C:\DKC3\Short Programming\VampireOut.txt

Input/Output examples on the next page.

Examples:

Input:

London Paris 12 6
London Paris 19 6
London Paris 24 5
London Munich 22 8
London Munich 19 8
London Milan 17 4
Paris Milan 19 9
Paris Munich 20 3
Milan Munich 20 4
Milan Madrid 24 6
London Madrid 0
*
Boston Philadelphia 24 4
Boston Pittsburgh 14 6
Boston Pittsburgh 20 6
Philadelphia Pittsburgh 2 2
Philadelphia Columbus 21 3
Pittsburgh Nashville 21 8
Pittsburgh Louisville 9 7
Pittsburgh Louisville 22 5
Louisville Nashville 4 2
Boston Nashville 0
*

Output:

London Paris Milan Madrid 20
2
Boston Pittsburgh Louisville Nashville 13
1

DKC^3 2020 – Short Programming Problems

9. Sorting Strings (5 points)

Given the ASCII values of each letter and number, write a program that takes in a string and sorts it in the following order:

1. Alphabetical
2. Uppercase letters before lowercase
3. All letters before numbers
4. Numbers in ascending order

Letter	ASCII Code	Letter	ASCII Code
a	097	A	065
b	098	B	066
c	099	C	067
d	100	D	068
e	101	E	069
f	102	F	070
g	103	G	071
h	104	H	072
i	105	I	073
j	106	J	074
k	107	K	075
l	108	L	076
m	109	M	077
n	110	N	078
o	111	O	079
p	112	P	080
q	113	Q	081
r	114	R	082
s	115	S	083
t	116	T	084
u	117	U	085
v	118	V	086
w	119	W	087
x	120	X	088
y	121	Y	089
z	122	Z	090

48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

Input/Output information on the next page.

Input
There will be ten test cases, each separated by an asterisk (*). Each test case will consist of a string containing alphanumeric characters, with no spaces.

Output
For each test case, output the sorted string. Lines of output should be separated by an asterisk (*).

Input File: D:\DKC3\Short Programming\SortingStringsIn.txt
Output File: D:\DKC3\Short Programming\SortingStringsOut.txt

Examples:

Input:

qdz3W71YRkp2t0FZLG8U
*
lwul69tzItBoNXEB4Yup
*

Output:

dFGkLpqRtUWYZz012378
*
BBEIllNopttuuwXYz469
*

10. Concrete Mix (10 points)

Ryan runs a basketball hoop installation service, and one of the steps in the installation process is to dig a square hole and fill it with concrete to offset the weight of the basketball hoop. Ryan has a tough time estimating the dimensions of the hole, as well as the amount of concrete mix needed for the installation, so he'd like you to write a program to help him out.

The width of the backboard and weight of the entire hoop is used to determine how wide and deep the square hole needs to be. Below are the formulas Ryan has come up with for determining the dimensions of the hole. When calculating the width of the hole, Ryan rounds up to the nearest whole inch as a safety precaution. When calculating the depth of the hole, he rounds down to the nearest whole inch.

$$\text{Width (inches)} = \frac{Weight\ of\ hoop\ (lbs)}{Width\ of\ backboard\ (in)} * 1.75$$

$$\text{Depth (inches)} = \frac{Weight\ of\ hoop\ (lbs)}{Width\ of\ hole\ (in)} + \frac{Width\ of\ backboard\ (in)}{2}$$

In addition to the hole dimensions, Ryan would also like to know how many 60lb bags of high-strength concrete mix his crew will need for each installation. Each 60lb bag of mix produces 0.45 cubic feet of concrete. Write a program that helps Ryan calculate how big the square hole needs to be, as well as how many bags of concrete he needs to complete each installation.

Input
There will be ten test cases, one per line. Each test case will be in the format "X, Y" where X is the weight of the hoop in pounds and Y is the width of the backboard in inches. Note that there will be a space following the comma in the input.

Output
The output for each test case should be on a new line, and needs to be in the format "A, B, C" where A is the width of the hole in inches, B is the depth of the hole in inches, and C is the number of bags of concrete mix needed. Note that there should be a space following each comma in the output.

Input File: C:\DKC3\Short Programming\ConcreteIn.txt
Output File: C:\DKC3\Short Programming\ConcreteOut.txt

Examples:

Input:
950, 72
675, 60

Output:
24, 75, 56
20, 63, 33

11. Missing Terms (15 points)

You will be given an equation of the form: **a + b = c**

Any one of the integer variables a, b or c will be missing. Your task is to find the missing term. Valid operators in the given equation will include: **+ - * / % ^**

Input
Each test case will be on its own line and consist of an equation with one of the variables replaced with a question mark. There are ten test cases.

Output
For each test case, output the integer that should replace the question mark to make the equation correct.

Input File: C:\DKC3\Short Programming\MissingTermsIn.txt
Output File: C:\DKC3\Short Programming\MissingTermsOut.txt

Examples:

Input:
2 + 6 = ?
? + 3 = 6

Output:
8
3

12. Frank's Fantastic Flapjacks (30 points)

Frank has always run the best pancake stand in the entire fair. He is so popular that he often runs out of ingredients, but he always tries to make as much money as he can before he does. Frank makes his famous flapjacks with six ingredients – his secret flour mix, buttermilk or almond milk, and at least one of three add-ins: chocolate chips, blueberries, and bacon.

He has some limits on his recipes too. Almond milk cannot be used for his bacon pancakes, and while blueberries can be mixed with chocolate and chocolate can be mixed with bacon, blueberries cannot be mixed with bacon. Each pancake must use one measure of flour mix, one measure of liquid, and one or more measures of mix-ins.

The prices for Frank's Flapjacks are calculated like so:

- Buttermilk Pancake = $2
- Almond Milk Pancake = $3
- Chocolate Chips = +$1
- Blue Berries = +$2
- Bacon = +$3

Frank will give you a list of six numbers representing his measures of flour, buttermilk, almond milk, chocolate chips, blueberries, and bacon in that order. Your job is to calculate the maximum amount of money Frank can make with those ingredients and return it as an integer.

Input
There are ten test cases, one per line. Each test case contains a sequence of six positive integers separated by a single space. The integers represent Frank's measures of flour, buttermilk, almond milk, chocolate chips, blueberries, and bacon.

Output
For each test case, output the maximum amount of money Frank can make in the form of an integer.

Input File: C:\DKC3\Short Programming\FranksFlapjacksIn.txt
Output File: C:\DKC3\Short Programming\FranksFlapjacksOut.txt

Examples:

Input:
1 1 0 1 1 1
6 4 2 4 2 2

Output:
5
28

13. Disk Space (10 points)

Write a program that will calculate which free blocks on a disk drive to use to download a file. The downloaded file must fit exactly into the free spaces on the disk. Write a program to help determine how to fit any sized file into the fewest free blocks possible.

Input
Your program input will consist of a list of numbers in a single line. The first number will be the size of the file to be downloaded in blocks. The following n numbers ($1 \leq n \leq 100$) represent the sizes of each available chunk of free space in blocks. Assume there are an unlimited number of blocks of each listed size. The number of blocks given may vary with each test case. There will be one test case per line. There are ten test cases.

Output
Output the list of block sizes used to download the file exactly. Output the block sizes in descending order. If there is no exact way to download the file, output 'no solution'.

Input File: C:\DKC3\Short Programming\DiskIn.txt
Output File: C:\DKC3\Short Programming\DiskOut.txt

Examples:

Input:
17 1 2 4 5
21 8 2 4

Output:
5 5 5 2
'no solution'

14. Extremely Arbitrary Strings (15 points)

In a code language, character-units are defined by a specific number of a character appearing in succession. Given a string in this language, you must determine the total number of distinct character-units in the string.

Input
Each input will be on its own line and consist of a string of characters with no spaces. Possible characters will be uppercase letters in the English alphabet or the digits 0-9. There are ten test cases.

Output
Each line of output should be a single number representing the total number of distinct character-units in the string.

Input File: C:\DKC3\Short Programming\ArbitraryIn.txt
Output File: C:\DKC3\Short Programming\ArbitraryOut.txt

Examples:

Input:
YYZYZZ
AAAA55AAAA

Output:
4
2

15. Rolling the Dice (25 points)

A group of people are playing a game that involves rolling various sets of dice. Each die has between 4 and 20 faces that are numbered from 1 to the number of faces. A set of dice can include up to ten dice, each with any number of faces. A move consists of rolling all the dice in a set simultaneously and summing the results to get a total score. For any given set of dice, the group want to know what the most likely score outcome is.

Your task is to read in a set of dice and output the most likely score from rolling them. If multiple scores are equally likely, list them in ascending order separated by a comma. Assume that this game takes place in a fictional universe where regular polyhedrons exist with any number of faces, so all dice are fair.

Input
Each input will be on its own line and consist of a list of dice in the set written as DN, where N is the number of faces. There will not be more than ten dice in a set and each die will have 4 to 20 faces. There are ten test cases.

Output
Each line of output should be a comma and space-separated list of the most likely score outcomes from rolling the set of dice.

Input Files: C:\DKC3\Short Programming\RollingIn.txt
Output Files: C:\DKC3\Short Programming\RollingOut.txt

Examples:

Input:
D8
D6 D6

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