DKC^3 2021 – 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 2021 – Short Programming Problems

1. Alphabetical Compression (5 points)

Write a program that takes in a string of characters, sorts them alphabetically, and then compresses them by adding the count of each character in front of it. If the letter appears 1 or 0 times in the string, you do not need to add the character count in front of it, just add the character if it appears once, and don't add it if it doesn't appear.

Input
Each test case will consist of a random string of characters that are all uppercase. There will be 10 test cases.

Output
The output will be a string that includes the count of a letter, followed by the letter itself. Letters should not repeat in the output.

Input File: C:\DKC3\Short Programming\AlphabeticalCompressionIn.txt
Output File: C:\DKC3\Short Programming\AlphabeticalCompressionOut.txt

Examples:

Input:
AAABAABBCDDDDFFEFE
BBBZZZIIIUIII

Output:
5A3BC4D2E3F
3B6IU3Z

DKC^3 2021 – Short Programming Problems

2. Scary Numbers (10 points)

Here is a worrying thought if you are triskaidekaphobic: many integers contain the pattern 13. There is only a single such integer between 1 and 100 (namely 13) but there are another 19 between 100 and 1,000 (such as 131 and 213). Between 1,000 and 10,000 there are 279 such numbers and there are 3671 between 10,000 and 100,000.

Note that we only count patterns if they are unbroken; for example, 103 does not contain the pattern 13. Given a scary pattern, write a program to determine how many integers in a given range contain that pattern.

Input
Each test case will consist of two lines: the first line will contain a single integer *n* indicating the scary pattern (1 <= *n* <= 99), the second line will contain two integers, *a* then *b*, (1 <= *a* < *b* <= 100000), indicating the range. There are 10 test cases.

Output
Your output should consist of a single number, indicating the number of integers between *a* and *b* (inclusive) that contain the pattern *n*.

Input File: C:\DKC3\Short Programming\ScaryIn.txt
Output File: C:\DKC3\Short Programming\ScaryOut.txt

Examples:

Input:
13
13 1350
1
2 20

Output:
84
10

DKC^3 2021 – Short Programming Problems

3. Mathematical Marvel (15 points)

Jimmy is a master of mathematics, in fact he can take any positive integer number and break it apart into an expression of A + B * C = D such that every number is different and the equation is true for a given D. Additionally, since Jimmy likes to keep things simple, use the smallest possible natural number values for which the equation is true such that A < B < C and the values for A and B are minimized in that order (i.e. A is the smallest possible value, then B is the smallest possible value).

Input
Each test case will consist of a single integer D. There are 10 test cases.

Output
Using the given value D, calculate and output the natural number values for A, B, and C on a single line, separated by spaces.

Input File: C:\DKC3\Short Programming\MathematicalMarvelIn.txt
Output File: C:\DKC3\Short Programming\MathematicalMarvelOut.txt

Examples:

Input:
25
872

Output:
1 2 12
1 13 67

DKC³ 2021 – Short Programming Problems

4. Euler's Number (20 points)

Euler's number (you may know it better as just *e*) has a special place in mathematics. While *e* can be calculated as a limit, there is a good approximation that can be made using discrete mathematics. The formula for *e* is:

$$e = \sum_{i=0}^{n} \frac{1}{i!}$$
$$= \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

Note that 0! = 1. Now as *n* approaches ∞, the series converges to *e*. When *n* is any positive constant, the formula serves as an approximation of the actual value of *e*. (For example, at *n* = 10 the approximation is already accurate to 7 decimals.)

You will be given a single input, a value of *n*, and your job is to compute the approximation of *e* for that value of *n*.

Input
Each test case will consist of a single integer *n*, ranging from 0 to 10,000. There are 10 test cases.

Output
For each test case, output a single real number – the approximation of *e* computed by the formula with the given *n*. All output must be accurate to an absolute or relative error of at most 10-12.

Input File: C:\DKC3\Short Programming\EulersIn.txt
Output File: C:\DKC3\Short Programming\EulersOut.txt

Examples:

Input:
3
15

Output:
2.666666666666
2.718281828458

DKC³ 2021 – Short Programming Problems

5. Grid Sum (25 points)

Given an 6x6 grid of single digit numbers, six-digit numbers can be read from each row, column and diagonal. Reading rows and diagonals from left to right, and columns from top to bottom, find the sum of all such numbers that can be formed from a given grid. If the same number occurs multiple ways in a grid, only add it once.

Input
Input for each test case will be a set of six lines of six numbers each with all numbers being a single digit integer in the range 1-9. Each test case will be separated by a new line. There are 10 test cases.

Output
For each test case, output a single integer representing the sum of all extracted six-digit numbers.

Input File: C:\DKC3\Short Programming\GridSumIn.txt
Output File: C:\DKC3\Short Programming\GridSumOut.txt

Examples:

Input:

1 2 3 4 5 6
1 2 3 4 5 6
6 5 4 3 2 1
6 5 4 3 2 1
1 1 1 1 1 1
9 9 9 9 9 9

3 3 3 3 3 3
3 3 3 3 3 3
3 3 3 3 3 3
2 2 2 2 2 2
2 2 2 2 2 2
2 2 2 2 2 2

Output:

5260776
1111110

6. Herding Sheep (20 points)

James the Shepherd needs to herd his flock of sheep into straight lines such that there are no empty spaces between sheep. He wants to accomplish this as quickly as possible so he can get home for dinner, and he needs your help.

The arrangement of the flock of sheep is represented as a string containing the characters '.' (period, representing an empty space) and '*' (asterisk, representing a sheep). In order to move a sheep, James needs to ensure the space next to the sheep exists and is empty before moving the sheep one space to the left or right. The flock is considered "lined up" when there are no empty spaces between sheep.

Your goal is to write a program that takes in the current arrangement of James' flock and outputs the minimum number of moves necessary for his sheep to line up.

Input
Each test case consists of a single line of input containing a string that represents the current arrangement of the flock of sheep (as stated above). There are 10 test cases.

Output
Each line of output should consist of a single integer representing the minimum number of moves needed to make the sheep line up.

Input File: C:\DKC3\Short Programming\SheepIn.txt
Output File: C:\DKC3\Short Programming\SheepOut.txt

Examples:

Input:
**.*..
....*.**

Output:
1
9

7. Password Generator (15 points)

You are working at a small paper company in "The Electric City". Your boss is not the brightest fish in the koi pond and keeps getting his accounts stolen because of the simple passwords he uses. This is resulting in him losing large sums of cash. So much so that he must settle for laptop batteries as gifts to a class he promised college tuition. At least the laptop batteries are lithium. As the "Assistant to The Regional Manager", you decide to assist him and create a password generator. You come up with a list of rules the generator should follow. Now, it is time for implementation.

Follow the guidelines below to produce a password generator for your boss, so he doesn't keep losing money.

- Step 1. The program should take in a list of five comma separated words.
  - The words have a minimum length of three characters.
  - The words have a maximum length of twenty characters.
  - The words will only consist of lower-case English characters.
- Step 2. Get the first, middle, and last letter of each word.
  - If the word doesn't have a letter perfectly in the middle take the middle right letter.
    - So, for the word "test", the middle letter would be "s".
- Step 3. Interleave each set of three letters from left to right.
  - So, if the five sets of three were: hjt,kde,tsl,pto,czq. The first 10 characters would be hktpcjdstz
- Step 4. Capitalize the first letter and every third letter after that.
  - So, hktpcjdstz would be HktPcjDstZ.
- Step 5. Insert a "$" symbol at the index equal to the number of vowels in the password string.
  - Consider the string to be zero-based. Meaning, the first character is at index zero.
  - You don't need to account for "sometimes y".
  - If there are no vowels in the password string after Step 4, add the "$" symbol to the beginning of the password string.
- Step 6. Append "DKC3" to the end of the password string.
- Step 7. Output the password string for your boss.

NOTE: The steps should be done in order and each step uses the string resulting from the previous steps.

If done correctly, your boss has a secure password, and all he needs to remember is the five words he used as input. You will surely receive a Dundie award for your efforts!

Input/Output examples on the next page.

Input
The input will consist of 10 test cases, each on its own line. Each test case will contain a string consisting of lowercase English characters and commas. There will be four commas that separate five words on each line. The length of the words will be between 3 and 20 characters.

Output
There should be one line of output for each test case. The output for each test case should be the password generated according to the guidelines.

Input File: C:\DKC3\Short Programming\PasswordGeneratorIn.txt
Output File: C:\DKC3\Short Programming\PasswordGeneratorOut.txt

Examples:

Input:
improvisation,family,salesman,parties,paper
ryan,jim,dwight,pam,phyllis

Output:
Ifs$PpiIstPnyNsrDKC3
Rjd$PpaIgaLnmTmsDKC3

8. Golfing (10 points)

When Kitty Forest goes golfing she always hits the ball the exact distance that is specified for the club she is using. She always hits the ball perfectly straight, and the distance from the tee to the hole is listed in feet in a straight line from the tee to the hole. Write a program that will list one of two things, either the fewest number of strokes it will take Kitty to get from the tee into the hole, or print "Can't be done.", if she can't get the ball into the hole.

Input
For each test case, the first number X on every line will be the distance from the tee to the hole in feet, where 1 <= X <= 10000. The next few numbers on the line will be the distances that her individual clubs will hit. There will be no more than 100 distances listed. There are 10 test cases.

Output
For each line of input, output the fewest number of strokes it will take Kitty to get from the tee into the hole, or print "Can't be done."

Input File: C:\DKC3\Short Programming\GolfIn.txt
Output File: C:\DKC3\Short Programming\GolfOut.txt

Examples:

Input:
504 4 80 100
503 4 80 100

Output:
6
Can't be done.

9. Reverse Polish Notation (20 points)

Reverse polish notation is a notation where the operators follow the operands (AKA Postfix). For example, "3 + 4" is written "3 4 +" in Reverse Polish Notation.

Write a program that reads in equations represented in Reverse Polish Notation and then outputs the solution to the equation.

Input
Each test case will consist of a single line, with a space separating each number/operator. Only the binary operators +, -, *, / will be used. There will be 10 test cases.

Output
For each test case, output a single integer representing the solution to the equation.

Input File: C:\DKC3\Short Programming\ReversePolishIn.txt
Output File: C:\DKC3\Short Programming\ReversePolishOut.txt

Examples:

Input:

3 4 * 2 + **This represents the equation: 3*4+2
2 3 4 + - **This represents the equation: 2-(3+4)

Output:

14
-5

10. Cake Tasting (15 points)

Fred and Wilma are getting married next month and they are excited to do some cake tasting to determine what kind of cake they will be serving at their reception. The baker has arranged n cakes on a circular table, numbering them 1 to n in a clockwise direction. She has provided Fred and Wilma with a list of the cake options she has for them to taste, and they quickly jot down which cakes they are interested in tasting and which cakes they are not interested in tasting.
To help them decide in which order to taste the cakes, Fred and Wilma come up with a tasting game. Below are the rules:

- At the very beginning of the game, if there is at least one cake on the table Fred and Wilma want to taste, they begin tasting Cake #1 whether they are interested in that cake or not.
- After tasting a cake, that cake is removed from the table and Fred and Wilma count ***k*** cakes clockwise beginning with the next one in the circle to find their next cake to taste.
    - If there are less than ***k*** cakes on the table, some cakes can participate in the count more than once and the last cake in the count is picked.
- If there are no cakes left on the table that Fred and Wilma are interested in tasting, the game ends.

Write a program that determines how many cakes Fred and Wilma tasted during their visit to the bakery.

Input/Output examples on the next page.

Input
Each test case consists of two lines of input. The first line of input contains two space-separated integers: *n* (representing the number of cakes on the table) and *k* (representing the count used for the tasting game). The second line of input contains a string of length *n* consisting of 1s and 0s. The 1s and 0s indicate which cakes Fred and Wilma are interested in trying, beginning clockwise from Cake #1. If Fred and Wilma want to try a cake, its respective position in the string will contain a 1, otherwise a 0. There are 10 test cases.

Output
Each line of output should consist of a single integer representing the number of cakes Fred and Wilma tasted at the bakery.

Input File: C:\DKC3\Short Programming\CakeIn.txt
Output File: C:\DKC3\Short Programming\CakeOut.txt

Examples:

Input:
6 4
000111
5 1
10011

Output:
4
5

11. Degrees of a Clock (10 points)

Given a time of day, determine the degree of the angle formed by the minute and hour hand of a standard clock.

Input
For each test case, the time will be given in a standard 24-hour format with a colon separating the hours and minutes. There are 10 test cases.

Output
For each line of input, output the smallest possible angle between the minute and hour hand.

Input File: C:\DKC3\Short Programming\DegreesOfAClockIn.txt
Output File: C:\DKC3\Short Programming\DegreesOfAClockOut.txt

Examples:

Input:
12:15
14:50

Output:
82.5
145

12. The Protracting Palindrome (25 points)

A palindrome is a word, number, phrase, or sequence of other characters which reads the same backward as forward, such as madam or racecar. A substring is a contiguous sequence of characters within a string. Given a string *s*, find the longest palindromic substring in *s*.

Input
The input will consist of 10 test cases, each on its own line. Each test case will be a string consisting of lowercase and/or uppercase English letters, numbers, and/or symbols. The possible symbols are: "!", "@", "#", "$", "%", "^", "&", "*". The length of each test case string *s* will be greater than 0 and less than or equal to 1000.

Output
There should be one line of output for each test case. Each line should consist of the longest palindrome substring from the given string *s*. If there is a tie, output the longest palindromic substring that is the furthest to the left. If there isn't a palindrome with a length greater than or equal to three in the string, output an underscore symbol.

Input File: C:\DKC3\Short Programming\ProtractingPalindromeIn.txt
Output File: C:\DKC3\Short Programming\ProtractingPalindromeOut.txt

Examples:

Input:
aa2
^atqk%@!#*@^!$&*#kj@$&$@jklnmopomn*#!&

Output:
a
kj@$&$@jk

13. Problematic Packing (35 points)

Tyler's shipping company always packs items into cube-shaped boxes with integer dimensions. Luckily, the things they pack are also always cube-shaped items with integer dimensions. Tyler always packs his boxes in a certain way – he always packs the largest item he can at any given time and proceeds until none of the remaining items will fit in the remaining space in the box. Given the dimensions of a box and of some number of items that need packing, your job is to determine the total volume of items Tyler packs into the box.

Input
Each test case will consist of a single line with set of integers separated by spaces. The first integer in the list will be the dimensions of the packing box, while all subsequent integers in the list will be dimensions of objects to pack. There are 10 test cases.

Output
For each test case, the output should consist of a number representing the total volume of all items packed into the box.

Input File: C:\DKC3\Short Programming\PackingIn.txt
Output File: C:\DKC3\Short Programming\PackingOut.txt

Examples:

Input:
6 2 2 2 3 3 3 3 3 5
5 8 1 1 1 1

Output:
125
4

DKC³ 2021 – Short Programming Problems

14. Candy Cypher (30 points)

Someone is stealing candy! To protect their candy stash, the two brothers have created a complex system with coded messages.

Percy (Spy Master)

- Picks daily key for secret messages
- Decides when to move the candy and where
- Encodes A-Z messages with the CoolCypher (see below) using the daily key
- Sends encoded messages to Bill in chat

Bill (Secret Agent)

- Memorizes key from Percy every day at midnight
- Reads Percy's messages using daily key and decodes them on paper
- Forgets to destroy decoded messages sometimes
- Always destroys messages before midnight

Sarah (Annoyed Suspect)

- Is suspected of candy thieving by her brothers but never stole their candy
- Is determined to find the true culprit
- Searches Bill's room often for forgotten messages
- Needs your help

You (Codebreaker)

- Sarah's friend who admins family chat server
- Have access to all encoded messages on server
- Have some decoded messages from Sarah
- Try to discover the key for Sarah so she can catch the real thief red-handed

Cool Cypher (Vigenère)

- The message "YOURCLOSET" can be coded with "KEY".
- So, Y + K becomes 25 + 11 = 36. Wrapping this around we get 9 or I.

- Repeating this for the other letters we get:

```
   YOURCLOSET
+  KEYKEYKEYK
   ISSBGJYWCD
```

Hints

- This is not a time travel problem
  - Percy and Bill must share the key first (at midnight)
  - Then Percy and Bill can send messages
  - Finally Sarah can find them and send to you

- For keys with repeated sections only the shortest possible unique key for the day's messages will be accepted
  - i.e. AAAAB AAC is a valid key
    Neither AAAB or AAABAACA will be accepted

Input/Output information on the next page.

Input
Your program takes two lines of input. The first line is all encoded messages from the server on a single day separated by commas. The second line is all the decoded messages Sarah has found that day, also separated by commas.
Example encoded message: { January 1, 2020 15:00:00, ISSBGJYWCD }
Example decoded message: { 1/1/20 3:02 PM, YOURCLOSET }

Output
The shortest possible daily key on a single line. Output a question mark instead if no key can be found.

Input File: C:\DKC3\Short Programming\CandyCypherIn.txt
Output File: C:\DKC3\Short Programming\CandyCypherOut.txt

Examples:

Input:

{ January 1, 2020 15:00:00, ISSBGJYWCD }
{ 1/1/20 3:02 PM, YOURCLOSET }
{ January 2, 2020 11:00:00, HRZOVKLFAN },{ January 2, 2020 13:00:00, LSQBWMPOZBEURV }
{ 1/2/20 11:30 AM, UNDERMYBED }

Output:

KEY
NEWKEY

15. Hearing Numbers (25 points)

Margaret is working on a new text-to-speech program, but is having some trouble getting the program to understand Latin numerals. As a workaround, she would like to use the fully functional Latin character section of the program, but needs some help to automatically translate the numeral into its spoken counterpart.

Input
Each test case will consist of a single line containing an English sentence with some numbers in it. There are 10 test cases.

Output
For each line of input, replace the numeric values with their word equivalent and output the adjusted sentence.

Input Files: C:\DKC3\Short Programming\HearingNumbersIn.txt
Output Files: C:\DKC3\Short Programming\HearingNumbersOut.txt

Examples:

Input:
I HAVE THE HIGH SCORE OF 100
42 IS THE ANSWER

Output:
I HAVE THE HIGH SCORE OF ONE HUNDERED
FOURTY TWO IS THE ANSWER