2015 International APL Problem Solving Competition – Phase I

Phase I Tips

- We have provided you with several test cases for each problem to help you validate your solution.
- We recommend that you build your solution using dfns. A dfn (direct function) is one or more APL statements enclosed in braces {}. The left hand argument, if any, is represented in a dfn by ⍺, while the right hand argument is represented by ⍵.
  Example:
  ```
        'Hello' {⍺,'-',⍵,'!'} 'world'
  Hello-world!
  ```
  A dfn terminates on the first statement that is not an assignment. If that statement produces a result, the dfn returns that value as its result.
  Example:
  ```
        'left' { ⍵ ⋄ ⍺ } 'right'
  right
  ```
  For more information on dfns, use the online help included with Dyalog or see page 152 in http://www.dyalog.com/MasteringDyalogAPL/MasteringDyalogAPL.pdf.
- The symbol ⍝ is the APL comment symbol. In some of the examples below, comments are provided to give more information.
- Some of the problem test cases use "boxed display" to make the structure of the returned results clearer. Boxing is enabled by default on www.TryAPL.org and can be enabled in Dyalog version 14.0 and later using the user command:
  ```
        ]box on
  ```
  Without boxed display enabled:
  ```
        ⍳¨⍳4
  1  1 2  1 2 3  1 2 3 4
  ```
  With boxed display enabled:
  ```
        ⍳¨⍳4
  ```
1	1 2	1 2 3	1 2 3 4

Phase I Problems

Sample Problem - I'd like to buy a vowel
Write a dfn to count the number of vowels in a character vector.

When passed the character vector 'APL Is Cool', your solution should return:
```
4
```

Below are 2 sample solutions. Both produce the correct answer, however the first solution would be ranked higher by the competition judging committee as it demonstrates better use of array oriented programming.

```
      {+/⍵∈'AEIOUaeiou'}'APL Is Cool' ⍝ better solution
4
      {(+/⍵='A')+(+/⍵='E')+(+/⍵='I')+(+/⍵='O')+(+/⍵='U')+(+/⍵='a')+
(+/⍵='e')+(+/⍵='I')+(+/⍵='o')+(+/⍵='u')}'APL Is Cool' ⍝ lesser solution
4
```

Problem 1 – Nag A Ram

Write a dfn that takes two character vectors as its left and right arguments and returns 1 if they are anagrams of each other. An anagram of a string uses all of the letters of the string ignoring word spacing, capitalisation, and punctuation.

Test cases:

```
      'anagram' {your_solution} 'Nag A Ram'
1

      'Dyalog APL' {your_solution} 'Dog Pay All'
1

      '' {your_solution} '  !#!'
1

      'abcde' {your_solution} 'zyxwvu'
0
```

Problem 2 – Longest Streak

Write a dfn that takes a numeric vector and returns the length of the longest streak of positive growth.

Test cases:

```
      {your_solution} 1 2 3 4 5 6 7 8 9
8

      {your_solution} 1
0

      {your_solution} 9 8 7 6 5 4
0

      {your_solution} 1 5 3 4 2 6 7 8
3
```

Problem 3 – Farey Tale

From http://en.wikipedia.org/wiki/Farey_sequence - In mathematics, the Farey sequence of order n is the sequence of completely reduced fractions between 0 and 1 which, when in lowest terms, have denominators less than or equal to n, arranged in order of increasing size. Each Farey sequence starts with the value 0, denoted by the fraction 0⁄1, and ends with the value 1, denoted by the fraction 1⁄1.

Write a dfn that takes an integer right argument and returns a vector of the terms in the Farey sequence of that order. Each element in the returned vector is itself a 2-element vector of numerator and denominator for the corresponding term.

Test cases (using boxed display):

```
      {your_solution} 0
```

0 1

```
      {your_solution} 1
```

0 1	1 1

```
      {your_solution} 5
```

0 1	1 5	1 4	1 3	2 5	1 2	3 5	2 3	3 4	4 5	1 1

Problem 4 – PDI – Progressive Dyadic Iota

The index-of function (X⍳Y) returns a simple integer array with the same shape as Y that identifies where the items of Y are first found in X. If an item of Y cannot be found in X, then the corresponding item of the returned array will be (1+⊃⍴X) - 1+ <the length of the leading dimension of X>.

Example:

```
     'DYALOG APL' ⍳  'AAALLLB'
3  3 3 4 4 4 11
```

Progressive dyadic iota is similar to ⍳ except that it returns the index of subsequent matches in the left argument until they are exhausted. Write a dfn that implements progressive dyadic iota.

Test cases:

```
      'DYALOG APL' {your_solution} 'AAALLLB'
3 8 11 4 10 11 11

      '' {your_solution} 'test'    ⍝ should work with empty left argument
1 1 1 1

      ⍬≡'test' {your_solution} ''  ⍝ should work with empty right argument
1
```

Problem 5 – He's so mean, he has no standard deviation.

The standard deviation of a population is calculated by taking the square root of the average of the squared differences of the values from their average value. The mathematical formula is:

$$\sqrt{\frac{\sum(x-\bar{x})^2}{n}}$$

Write a dfn that returns the population standard deviation of its numeric array right argument.

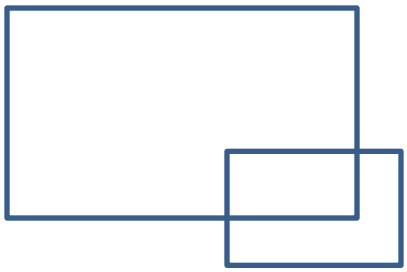
Test cases:

```
      {your_solution} 1 2 3 4 5
1.414213562

      {your_solution} 10 10⍴⍳100  ⍝ and higher dimensions as well
28.86607005
```

Problem 6 – Hey! This is My Space

Write a dfn that takes, as each of its right and left arguments, a pair of co-ordinates representing diagonally opposite corners of two rectangles and returns a 1 if the rectangles overlap. The co-ordinates could be either (upper-left, lower-right) or (upper-right, lower left).



Test cases:

```
      (1 1)(5 5) {your_solution} (3 3)(7 7)
1

      (1 1)(5 5) {your_solution} (5 5)(1 1)
1

      (1 1)(3 3) {your_solution} (4 4)(7 7)
0

      (1.1 1.1)(5.5 5.5) {your_solution} (3.3 3.3)(4.4 4.4)
1
```

Problem 7 – Just In (Upper) Case

There are many ways to convert text case in APL. The following function uses Dyalog's built-in regular expression support to convert a string to uppercase:

```
      ucf←{⊃('.+' ⎕S '\u0')⍵}
```

Example:

```
      ucf 'This is neat'
THIS IS NEAT
```

In many instances, it is desirable to perform case-insensitive comparisons and operations. Write a dop (direct operator) with the following syntax:

```
      left_arg (ucf {your_solution} fn) right_arg
```

where:

- `ucf` is the uppercase function (you can use the one above or construct your own)
- `fn` is a primitive or user-defined dyadic function that operates on character data
- left_arg is the left argument
- `right_arg` is the right argument

The dop will execute fn in a case-insensitive manner on the left and right arguments.

Test cases:

```
      'dyalog'( ucf {your_solution} ⍳ )'APL'
3 7 4

      'bramley'( ucf {your_solution} ∈ )'HAMPSHIRE'
0 1 1 1 0 1 0
```

Problem 8 – Unlucky 13

You've just been hired by a new company and you've learned that the owner has triskaidekaphobia (fear of the number 13). As such he's issued a corporate mandate that no output from any program shall include the number 13 – 12.99 will be used instead. Your boss wants you to implement a dfn to process output and change the 13s to 12.99. For now, you need only concern yourself with numeric data and not worry about 13s that are formatted in text.

Test cases:

```
      {your_solution} 13
12.99

      {your_solution} ⍳15
1 2 3 4 5 6 7 8 9 10 11 12 12.99 14 15

      {your_solution} 6+5 6 7 8 9
11 12 12.99 14 15

      {your_solution} 13 130 13.13 1300  ⍝ we only care about the number 13
12.99 130 13.13 1300
```

Problem 9 – I'd Like Mine Scrambled Please

There is an urban myth[1] about some research that was supposedly done at Cambridge University that purports changing the order of the interior letters in the words of a sentence does not significantly hamper the readability of the sentence.

Put another way...

Teher is an ubran mtyh aobut smoe rseaecrh taht was spuopesldy dnoe at Cmarbdige Uinevsrtiy taht pruoptrs cahgnnig the odrer of the itnreoir lteetrs in the wrods of a sneetcne deos not sgiinifactnly hmaepr the raeadibilty of the sneetcne.

[1] http://www.foxnews.com/story/2009/03/31/if-can-raed-tihs-msut-be-raelly-smrat/

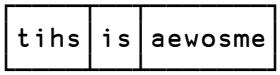
In the above example, for words that contain more than 3 letters, the first and last letters remain the same while all of the interior letters are transposed in groups of two, ignoring punctuation. If there are an odd number of letters in the interior of the word, then the last letter is left as it is.

Write a dfn that takes a character vector word as its left argument and returns the word's letters juxtaposed as described above.

Test cases:

```
      {your_solution} 'argument'
agrmunet

      ]box on
      {your_solution}¨'this' 'is' 'awesome'
```




```
      ]box off
      {your_solution}¨'this' 'is' 'awesome'
 tihs  is  aewosme
```

Problem 10 – Blaise'ing a Trail

The APL expression to compute Pascal's triangle of order n is fairly simple.

```
      {{⍵∘.!⍵}0,⍳⍵} 10
1  0  0   0   0   0   0   0  0  0 0
1  1  0   0   0   0   0   0  0  0 0
1  2  1   0   0   0   0   0  0  0 0
1  3  3   1   0   0   0   0  0  0 0
1  4  6   4   1   0   0   0  0  0 0
1  5 10  10   5   1   0   0  0  0 0
1  6 15  20  15   6   1   0  0  0 0
1  7 21  35  35  21   7   1  0  0 0
1  8 28  56  70  56  28   8  1  0 0
1  9 36  84 126 126  84  36  9  1 0
1 10 45 120 210 252 210 120 45 10 1
```

However, the output leaves something to be desired. Write a dfn that takes an integer right argument representing the order of the Pascal's triangle to be created and returns a "nicely" formatted character matrix. Because we're using a mono-spaced font and can only format whole character positions, lines to not have to be perfectly centered and may vary by a character position.

Test cases:

```
      {your_solution} 10
                  1
                1 1
               1 2 1
              1 3 3 1
             1 4 6 4 1
           1 5 10 10 5 1
         1 6 15 20 15 6 1
        1 7 21 35 35 21 7 1
       1 8 28 56 70 56 28 8 1
     1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

      {your_solution} 1
 1
1 1

    {your_solution} 0
1
```