APL Problem Solving Competition Phase I

The following shows what a typical Phase I problem description looks like. It also presents some possible solutions of varying quality, and explains how to provide your own solution. Content that doesn't appear in regular Phase I problems is formatted like this paragraph.

Each problem begins with a task description, followed by a hint suggesting one or more APL primitives. These may be helpful in solving the problem, but you are under no obligation to use them. Clicking on a primitive in the hint will open the Dyalog documentation page for the suggested primitive.

After the hint is a section of example cases which, among others, are included in the automated tests. Use these as a basis for implementing your solution.

Sample: Counting Vowels A

Write an APL function to count the number of vowels (A, E, I, O, U) in an array consisting of uppercase letters (A–Z).

Hint: The membership function `X∈Y` could be helpful for this problem.

Examples

```
      (fn) 'COOLAPL'
3
      (fn) ''    ⍝ empty argument
0
      (fn) 'NVWLSHR'    ⍝ no vowels here
0
```

Below are three sample solutions. All three produce the correct answer, but the first two functions would be ranked higher by the competition judging committee. This is because the first two demonstrate better use of array-oriented programming.

```
      ({+/⍵∈'AEIOU'}) 'COOLAPL'    ⍝ good dfn
3
      (+/∈∘'AEIOU') 'COOLAPL'    ⍝ good tacit function
3
      ⍝ suboptimal dfn:
      {(+/⍵='A')+(+/⍵='E')+(+/⍵='I')+(+/⍵='O')+(+/⍵='U')} 'COOLAPL'
3
```

2020 Phase I Problem Set

1: Let's Split!

Write a function that, given a right argument `Y` which is a scalar or a non-empty vector and a left argument `X` which is a single non-zero integer so that its absolute value is less or equal to `≢Y`, splits `Y` into a vector of two vectors according to `X`, as follows:

- If `X>0`, the first vector contains the first `X` elements of `Y` and the second vector contains the remaining elements.
- If `X<0`, the second vector contains the last `|X` elements of `Y` and the first vector contains the remaining elements.

💡 **Hint:** The *Take* function `X↑Y` might be useful for this problem.

Examples

```
      9 (fn) 'SplittingHairs'   ⍝ using ]Boxing on
```

Splitting	Hairs

```
      ¯3 (fn) 'DyalogAPL'
```

Dyalog	APL

```
      10 (fn) ⍳10
```

1 2 3 4 5 6 7 8 9 10	

```
      1 (fn) 'works' 'with' 'words' 'also'
```

works	with words also

2: Character Building

UTF-8 encodes Unicode characters using 1-4 integers for each character. Dyalog APL includes a system function, ⎕UCS, that can convert characters into integers and integers into characters. The expression `'UTF-8'⎕UCS` converts between characters and UTF-8.

Consider the following:

```
      'UTF-8'⎕UCS 'D¥α⌊○9'
68 194 165 226 141 186 226 140 138 226 151 139 57
      'UTF-8'⎕UCS 68 194 165 226 141 186 226 140 138 226 151 139 57
D¥α⌊○9
```

How many integers does each character use?

```
      'UTF-8'⎕UCS¨ 'D¥α⌊○9' ⍝ using ]Boxing on
```

68	194 165	226 141 186	226 140 138	226 151 139	57

The rule is that an integer in the range 128 to 191 (inclusive) continues the character of the previous integer (which may itself be a continuation). With that in mind, write a function that, given a right argument which is a simple integer vector representing valid UTF-8 text, encloses each sequence of integers that represent a single character, like the result of `'UTF-8'⎕UCS¨'UTF-8'⎕UCS` but does not use any system functions (names beginning with ⎕)

💡 **Hint:** Use ⎕UCS to verify your solution.

Examples

```
      (fn) 68 194 165 226 141 186 226 140 138 240 159 148 178 57   ⍝
using ]Boxing on
```

68	194 165	226 141 186	226 140 138	240 159 148 178	57

```
      (fn) 68 121 97 108 111 103   ⍝ 'Dyalog'
```

68	121	97	108	111	103

```
      (fn) ⍬   ⍝ '' (any empty vector result is acceptable here)
```

3: Excel-lent Columns ◫

A Microsoft Excel spreadsheet numbers its rows counting up from 1. However Excel's columns are labelled alphabetically — beginning with A–Z, then AA–AZ, BA–BZ, up to ZA–ZZ, then AAA–AAZ and so on.

Write a function that, given a right argument which is a character scalar or non-empty vector representing a valid character Excel column identifier between A and XFD, returns the corresponding column number

💡 **Hint:** The *Decode* function `X⊥Y`.

Examples

```
      (fn) 'A'
1

      (fn) 'APL'
1104
```

4: Take a Leap 📅

Write a function that, given a right argument which is an integer array of year numbers greater than or equal to 1752 and less than 4000, returns a result of the same shape as the right argument where 1 indicates that the corresponding year is a leap year (0 otherwise).

A leap year algorithm can be found here.

💡 **Hint:** The *Residue* function `X|Y` and the *Outer Product* operator `∘.` could be useful for this problem.

Examples

```
      (fn) 2020
1

      (fn) ⍬   ⍝ returns an empty vector

      (fn) 1900+10 10⍴⍳100
0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 0 0
0 1 0 0 0 1 0 0 0 1
```

5: Stepping in the Proper Direction 👣

Write a function that, given a right argument of 2 integers, returns a vector of the integers from the first element of the right argument to the second, inclusively.

💡 **Hint:** The *Index Generator* function `⍳⍴` function could be useful when solving this problem.

Examples

```
      (fn) 3 10
3 4 5 6 7 8 9 10

      (fn) 4 ¯3
4 3 2 1 0 ¯1 ¯2 ¯3

      □←r←(fn) 42 42
42

      ⍴r   ⍝ this is also a vector
1
```

6: Please Move to the Front ↩️

Write a function that, given a right argument which is an integer vector and a left argument which is an integer scalar, reorders the right argument so any elements equal to the left argument come first while all other elements keep their order.

💡 **Hint:** The *Grade Up* function `⍋⍴` could be helpful for this problem.

Examples

```
      3 (fn) 1 2 3 4 1 3 1 4 5
3 3 1 2 4 1 1 4 5

      3 (fn) ,1   ⍝ the , makes 1 into a vector
1

      42 (fn) ⍬   ⍝ empty right argument gives empty result
```

7: See You in a Bit

A common technique for encoding a set of on/off states is to use a value of 2^n for the state in position *n* (origin 0), 1 if the state is "on" or 0 for "off" and then add the values. Dyalog APL's component file permission codes are an example of this. For example, if you wanted to grant permissions for read (access code 1), append (access code 8) and rename (access code 128) then the resulting code would be 137 because that's 1 + 8 + 128.

Write a function that, given a non-negative right argument which is an integer scalar representing the encoded state and a left argument which is an integer scalar representing the encoded state settings that you want to query, returns 1 if all of the codes in the left argument are found in the right argument (0 otherwise).

💡 **Hint:** The *Decode* function `X⊥Y` and the derived *Inverse* operator `⍣¯1` could be helpful for decoding the states.

Examples

```
      2 (fn) 7   ⍝ is 2 in 7 (1+2+4)?
1

      4 (fn) 11   ⍝ is 4 in 11 (1+2+8)?
0

      3 (fn) 11   ⍝ is 3 (1+2) in 11 (1+2+8)?
1

      4 (fn) 0   ⍝ is 4 in 0?
0
```

8: Zigzag Numbers

A zigzag number is an integer in which the difference in magnitude of each pair of consecutive digits alternates from positive to negative or negative to positive.

Write a function that takes a single integer greater than or equal to 100 and less than 10^{15} as its right argument and returns a 1 if the integer is a zigzag number, 0 otherwise.

💡 **Hint:** Your solution might make use of *N-wise Reduction* `X f/ Y`.

Examples

```
      (fn) 123
0

      (fn) 132
1

      (fn) 31115
0

      (fn) 3141514131415
1
```

9: Rise and Fall ⛰️

Write a function that, given a right argument which is an integer scalar or vector, returns a 1 if the values of the right argument conform to the following pattern (0 otherwise):

- The elements increase or stay the same until the "apex" (highest value) is reached
- After the apex, any remaining values decrease or remain the same

💡 **Hint:** The *Maximum* function `X⌈Y` combined with the *Reduce* operator `f/Y` and the *Reverse* function `⌽Y` can help with solving this problem.

Examples

```
      (fn) 1 3 3 4 5 2 1
1

      (fn) 42
1

      (fn) 1 3 2 4
0

      (fn) 23 23 23
1

      (fn) ⍬  ⍝ empty vector
1
```

10: Stacking It Up

Write a function that takes as its right argument a vector of simple arrays of rank 2 or less (scalar, vector, or matrix). Each simple array will consist of either non-negative integers or printable ASCII characters. The function must return a simple character array that displays identically to what `{⍞←⍵}¨` displays when applied to the right argument.

💡 **Hint:** The *Mix* `↑⍤1`, *Split* `↓⍤1`, and *Format* `⍕⍤1` functions could be helpful for solving this problem.

Examples

All results will look identical with `]Boxing on` as they are simple (non-nested) character arrays.

```
      (fn) 'Hi' 'Earth'
Hi
Earth

      (fn) (3 3⍴⍳9)(↑'Adam' 'Michael')(⍳10) '*'(5 5⍴⍳25)
1 2 3
4 5 6
7 8 9
Adam
Michael
1 2 3 4 5 6 7 8 9 10
*
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25

      (fn) 'O' 'my!'
O
my!

      (fn) ,⊂⍳4
1 2 3 4

      (fn) ,'A'
A
```