2020 APL Problem Solving Competition – Phase II Problem Descriptions

Errata

This section will contain descriptions of any corrections that are made to this document after its initial release. When such corrections are made, newer versions of this document and the **Contest2020.zip** file will be uploaded to the contest website. If you have registered for the competition, then we will notify you by email when updated Phase II materials are made available.

11 May 2020

The following amendments were made:

- Problem 6 – Merge had a discrepancy in the examples regarding the use of "lastname" and "surname". The **merge1.json** and **merge2.json** files in the **Data** folder of the **Contest2020.zip** file incorrectly had "surname" instead of "lastname". These files have been corrected and the example amended from:

```
      ⊢merge1←⊃⎕NGET '/Data/merge1.json'
{
    "firstname":"Drake",
    "surname":"Mallard",
    "prize":"yoyo",
    "value":100
}
      ns←⎕JSON merge1   ⍝ convert JSON to namespace
      ns.⎕NL ¯2         ⍝ variables in the namespace
┌─────────┬─────┬───────┬─────┐
│firstname│prize│surname│value│
└─────────┴─────┴───────┴─────┘
```

to:

```
      ⊢merge1←⊃⎕NGET '/Data/merge1.json'
{
    "firstname":"Drake",
    "lastname":"Mallard",
    "prize":"yoyo",
    "value":100
}
      ns←⎕JSON merge1   ⍝ convert JSON to namespace
      ns.⎕NL ¯2         ⍝ variables in the namespace
┌─────────┬─────┬────────┬─────┐
│firstname│prize│lastname│value│
└─────────┴─────┴────────┴─────┘
```

26 May 2020

The following amendments were made:

- Problem 8 – The first two examples incorrectly had the function name as `Parts`. The examples have been amended from:

```
Parts 1 2 3 4 5 ⍝ odd total?  Can't split evenly, so return ⍬

Parts 1 3 ⍝ can't split evenly, so return ⍬
```

to:

```
Balance 1 2 3 4 5 ⍝ odd total?  Can't split evenly, so return ⍬

Balance 1 3 ⍝ can't split evenly, so return ⍬
```

Welcome!

This year's Phase II problems cover multiple domains, including bioinformatics, finance, information retrieval, algorithm development, text processing, and recreational computing. Each problem has been assigned a difficulty level – low, medium or hard – based on the judges' impressions.

Unlike previous years, you are not required to complete any specific problems. You must solve at least 1 (one) problem to be considered for a prize; if a problem has more than one task, then you need to solve all its tasks for the problem to be considered solved. We encourage you to solve as many problems as possible. The judging committee takes the following criteria into consideration:

- Did you solve the problem? Working solutions rate higher than non-working ones. 😊
- The difficulty of the problem. If two entrants solve two different problems equally well, the entrant who solves the more difficult problem will likely receive higher consideration.
- Does your solution demonstrate an understanding of the problem, including the implications imposed by edge cases? Is it well commented, but without writing a novel? Does the code flow smoothly?
- The appropriate application of array-oriented thinking. One of APL's strengths is its ability to process entire arrays at once rather than having you write code to loop over each element. However, not every problem has an elegant array-oriented solution. We look for your ability to discern where an array-oriented approach is beneficial.
- Brevity is not a primary consideration. This is not a code-golfing contest.
- Performance is a consideration. If two solutions are comparable in every other aspect, the faster one will likely receive higher consideration.
- Again, we encourage you to solve as many problems as possible. If two entries are comparable in every other aspect, the one that solves more problems will likely receive higher consideration.

Good Luck and Happy Problem Solving!

Note

Some of the examples are displayed using the user command setting `]Boxing on` to more clearly depict the structure of the displayed data. For example:

```
      ('Dyalog' 'APL')(4 4⍴⍳16) 5
 Dyalog  APL     1  2  3  4  5
                 5  6  7  8
                 9 10 11 12
                13 14 15 16

      ]Boxing on
Was OFF
      ('Dyalog' 'APL')(4 4⍴⍳16) 5
┌────────────┬───────────┬─┐
│┌──────┬───┐│ 1  2  3  4│5│
││Dyalog│APL││ 5  6  7  8│ │
│└──────┴───┘│ 9 10 11 12│ │
│            │13 14 15 16│ │
└────────────┴───────────┴─┘
```

How to Participate in Phase II

You'll need to download the file named Contest2020.zip from the competition website https://dyalogaplcompetition.com. This file contains several files including:

- This document (which is also downloadable separately).
- Contest2020.dws – the contest workspace to use if you want to use Dyalog APL's development environment.
- Contest2020.dyalog – a template namespace script that contains syntactically-correct stub functions for all Phase II problems. This allows you to use your preferred editor to build your solutions. If you happen to like Visual Studio Code, there are extensions from Optima Systems Ltd. to help with editing, syntax coloring, etc. (Search the Visual Studio Code Extensions Marketplace for "OptimaSystems".)
- Any data files necessary to solve the problems.

Develop your solutions using either the workspace (if using Dyalog's development environment) or the template (if using an external editor).
Your solutions should consist of only functions or operators and not depend on any global variables. You may write other functions to organize your code as you deem necessary.

Using the Contest2020.dws workspace

The **Contest2020.dws** workspace requires Dyalog APL v17.0 or later. If you use an earlier version of Dyalog APL, you should use the **Contest2020.dyalog** template file described below. The workspace contains:

- **`#.Problems`** – a namespace in which you will develop your solutions. **`#.Problems`** contains:
  - syntactically correct stubs for all of the functions described in the problem descriptions. These function stubs are implemented as traditional APL functions (tradfns) but you can change the implementation to dfns if you prefer; either form is acceptable.
  - any sample data elements mentioned in the problem descriptions.

  Any sub-functions you develop as a part of your solution should be located in **`#.Problems`**
- **`#.SubmitMe`** – a function used to package your solution for submission. On Microsoft Windows this function will display a GUI form to allow you to enter a description of yourself and any feedback on the competition. On non-Windows platforms, you'll be presented with a character-based prompt and response interaction.

Important! Make sure you save your work using the `)SAVE` system command!

Once you are ready to submit your solutions, run the **`#.SubmitMe`** function, enter the requested information and click the **Save** button. **`#.SubmitMe`** will create a file called **Contest2020.dyalog** which will contain any code or data you placed in the **`#.Problems`** namespace. You will then upload the **Contest2020.dyalog** file using the contest website.

Using the Contest2020.dyalog template file

This file contains the correct structure for submission. You can populate it with your code, but do not change the namespace structure. Once you have developed your solution, edit the `AboutMe` and `Reaction` variable definitions at the top of the file and upload the file using the contest website.

If you use a non-Dyalog APL system to develop your solutions, they will nonetheless need to execute under Dyalog APL. Therefore your solution should only use APL features that are common between your APL system and Dyalog APL.

If use Dyalog APL, including versions prior to v17.0, you can load the **Contest2020.dyalog** file into your session using the `]load` user command. For instance, assuming you've unzipped the zip file into a folder named **/contest/**, you would use

```
      ]load /contest/Contest2020
```

Use of tacit or derived functions in Phase II

If you use the **`Contest2020.dws`** workspace and you want to use tacit or derived functions in your Phase II solutions, they need to either:

1. be defined inline within a tradfn or dfn. For example:

```
    ∇ r←Foo w;Avg
[1]    Avg←+/÷≢
[2]    ...
     ∇
```

or

```
Goo←{
     Avg←+⌿÷≢
     ...
}
```

2. be "wrapped" in a tradfn. For example:

```
     ∇ Avg←Avg
[1]    Avg←+⌿÷≢
     ∇
```

If you use the **`Contest2020.dyalog`** template file, you may enter tacit or derived functions directly in the file.

Problem 1 – Take a Dive (1 task)
Level of Difficulty: Low

In an international competition like the Olympics, a diver's scores are determined by their execution of the dive and the degree of difficulty of the dive. The degree of difficulty for a particular dive is a number greater than 1 and is calculated on the basis of the dive's body position, the number of somersaults and/or twists, and other factors. Currently the most difficult dive, a reverse 4.5 somersault in pike position, has a 4.8 degree of difficulty.

Scores are calculated as follows:

1. Each judge rates the execution of the dive from 0 ("completely failed") to 10 ("excellent")
2. Outliers are removed depending on how many judges are used:
    - For seven judges, the lowest two and highest two scores are discarded
    - For five judges, the lowest and highest scores are discarded
    - For three judges, all scores are used
3. The remaining scores are totaled.
4. The total is multiplied by the degree of difficulty.

Task 1: Write a function named **DiveScore** that has the following syntax:

```
      score ← dd DiveScore scores
```

where:

- the left argument **dd** is a single number representing the degree of difficulty.
- the right argument **scores** is a numeric vector of length 3, 5, or 7 representing the judges' scores. Your code does not need to check that the length of **scores** is 3, 5, or 7; you may assume the argument will always be a correct length.
- the result **score** is the score computed using the steps above rounded to at most 2 decimal places.

Examples:

```
      2.8 DiveScore 7 7.5 6.5 8 8 7.5 7
61.6

      2.9 2.6 2.7 DiveScore¨(7 7.5 6.5 8 8 7.5 7)(9.5 8 8.5)(7.5 7 7 8.5 8)
63.8 67.6 60.75
```

Problem 2 – Another Step in the Proper Direction (1 task)
Level of Difficulty: Medium

In Problem 5 of Phase 1, "Stepping in the Proper Direction", you were asked to write an expression 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. This problem enhances that functionality.

Task 1: Write a function named `Steps` which has the following syntax:

```
      steps ← {p} Steps fromTo
```

where:

- the right argument `fromTo` is a 2-element integer vector representing the start and end points of the result.
- the optional left argument **p** is a single number as follows:
  - If **p** is negative, its absolute value represents the number of equally sized steps to take.
    - If **p** is not an integer, treat it like `⌊p`. (e.g. `¯4.7` is treated as `¯5`)
  - If **p** is positive, it represents the step size.
    - If **p** does not evenly divide `fromTo`, the last step will be `<p`.
  - If **p** is 0, return the first element of `fromTo` (as if you took zero steps)
  - If **p** is not provided, `Steps` should behave exactly as Problem 5 of Phase 1.
    See default left argument if you're using a dfn or defined functions if you're using a tradfn.
- the result, `steps`, is a numeric vector such that `fromTo≡(⊣/,⊢/)steps` and whose interior elements, if any, represent intermediate points.

Examples:

```
      Steps 4 ¯3 ⍝ same as Phase 1 Problem 5
4 3 2 1 0 ¯1 ¯2 ¯3

      1 Steps 4 ¯3 ⍝ step size of 1
4 3 2 1 0 ¯1 ¯2 ¯3

      1.5 Steps ¯3 4 ⍝ step size of 1.5 (note last step is size 1)
¯3 ¯1.5 0 1.5 3 4

      1.5 Steps 4 ¯3 ⍝ step size of 1.5 (note last step is size 1)
4 2.5 1 ¯0.5 ¯2 ¯3

      ¯4 Steps 42 42 ⍝ 4 zero-sized steps
42 42 42 42 42

      4⍕¯7 Steps 3 6  ⍝ result formatted to 4 decimal places by using 4⍕
 3.0000 3.4286 3.8571 4.2857 4.7143 5.1429 5.5714 6.0000
```

Problem 3 – Past Tasks Blast (1 task)
Level of Difficulty: Medium

We're going to do a little web scraping in this problem. Dyalog Ltd has made links to PDF files containing problem sets from past competitions available on `https://www.dyalog.com/student-competition.htm`.

Task 1: Write a function named **`PastTasks`** which has the following syntax:

```
    urls ← PastTasks url
```

where:

- the right argument, **`url`**, is the character vector `'https://www.dyalog.com/student-competition.htm'`.
- the result, **`urls`**, is a vector of character vectors each representing a fully qualified URL for a PDF file linked on the web page. The order of the URLs is not significant.

Notes:

- We can recommend a couple of ways to retrieve the content of the web page.
    - Use the Dyalog utility `HttpCommand`. You should load `HttpCommand` manually outside of your function (i.e. your function does not need to load `HttpCommand`). `HttpCommand.Documentation` will return documentation for `HttpCommand`.
      ```
          ]load HttpCommand
      ```
      Then in your function, call `HttpCommand`.
      ```
      r←HttpCommand.Get 'https://www.dyalog.com/student-competition.htm'
      ```
      `HttpCommand` returns a namespace. The payload of the response is found in the `Data` element
      ```
      pageContent←r.Data
      ```
    - Use `⎕SH` and the curl command (if it's available on your operating system)
      ```
      pageContent←∊⎕SH 'curl https://www.dyalog.com/student-competition.htm'
      ```
- There are several ways to parse the page content. A few of them are:
    - Use `⎕XML` to convert the HTML to matrix form (the page is XHTML-compliant and therefore parseable by `⎕XML`).
    - Use regular expressions with with the regex search system function `⎕S`.
    - Use APL's search primitive functions
    - Some combination of the above
- The links to the PDF files are found in the `href` attribute of `<a>` (anchor) elements. These links are relative to the base address for the page, which is found in the `href` attribute of the `<base>` element.

Example: your function should return all the URLs listed below, though not necessarily in the same order

```
      ≢urls ← PastTasks 'https://www.dyalog.com/student-competition.htm'
18
      ↑8↑urls  ⍝ the first 8 URLs
https://www.dyalog.com/uploads/files/student_competition/2019_problems_phase1.pdf
https://www.dyalog.com/uploads/files/student_competition/2019_problems_phase2.pdf
https://www.dyalog.com/uploads/files/student_competition/2018_problems_phase1.pdf
https://www.dyalog.com/uploads/files/student_competition/2018_problems_phase2.pdf
https://www.dyalog.com/uploads/files/student_competition/2017_problems_phase1.pdf
https://www.dyalog.com/uploads/files/student_competition/2017_problems_phase2.pdf
https://www.dyalog.com/uploads/files/student_competition/2016_problems_phase1.pdf
https://www.dyalog.com/uploads/files/student_competition/2016_problems_phase2.pdf
```

Problem 4 – Bioinformatics (2 tasks)
Level of Difficulty: Medium

For many years we have used problems from Rosalind inour competition. Rosalind is an excellent bioinformatics learning platform. This year we've selected two problems for you to solve.

Task 1: Write a function named **`revp`** which solves the Location Restriction Sites problem found at http://rosalind.info/problems/revp/. **`revp`** finds all the reverse palindromes of length between 4 and 12 in the input DNA string.

Note: What's a reverse palindrome in genetics?
The complement of a DNA string swaps 'A' for 'T', 'T' for 'A', 'C' for 'G', and 'G' for 'C'. 'ACTG' is the complement of 'TGAC'. 'GTCA' is the reverse complement of 'TGAC'. A reverse palindrome is a section of DNA that matches the reverse of its complement.

`revp` has the following syntax:

```
      r ← revp dna
```

where:

- the argument **`dna`** is a character vector representing a DNA string.
- the result, **`r`**, is a 2-column numeric matrix of the [;1] positions [;2] lengths of the reverse palindromes.

Notes:

- Two sets of sample dataset and output files are found in the /Data/ folder of the contest zip file.
  rosalind_revp_1_dataset.txt / rosalind_revp_1_output.txt
  rosalind_revp_2_dataset.txt / rosalind_revp_2_output.txt
- To read a dataset from the file, you can use the following (replacing `filename` with the name of the actual file):
  ```
  dataset←∊1↓⊃⎕NGET 'filename' 1
  ```
- As noted on the problem webpage, the result can be in any order.
- You can check your solution by submitting it on Rosalind.

Examples:

```
      revp 'TCAATGCATGCGGGTCTATATGCAT'
 5 4
 7 4
17 4
18 4
21 4
 4 6
 6 6
20 6

      ≢revp ∊1↓⊃⎕NGET 'rosalind_revp_1_dataset.txt' 1
92

      ≢revp ∊1↓⊃⎕NGET 'rosalind_revp_2_dataset.txt' 1
89
```

Task 2: Write a function named **`sset`** which solves the Counting Subsets problem found at http://rosalind.info/problems/sset/. **`sset`** has the following syntax:

```
      r ← sset n
```

where:

- the argument, **`n`**, is a positive integer ≤1000.
- the result, **`r`**, is an integer of the total number of subsets that can be made from a set of **`n`** elements modulo 1,000,000.

Notes:

- Dyalog APL does not have unlimited length integers, so the main trick to this task is understanding modular multiplication.
- You can also check your solution by submitting it on Rosalind.

Examples:

```
      sset 3
8

      sset 857
551872

      sset 870
935424
```

Problem 5 – Future and Present Value (2 tasks)
Level of Difficulty: Medium

¥ $
€ £

Future value is the value of a current asset at a future date based on an assumed rate of return. If the rate of return is constant, the future value can be calculated as:

$$\boldsymbol{I} \times (\mathbf{1} + \boldsymbol{R})^{\boldsymbol{T}}$$

In APL: **`I×(1+R)*T`**

where:

- I is the current value
- R is the rate of return
- T is the number of terms, generally years, into the future

If the rate of return is not constant then you need to successively multiply by each future rate.

```
      investment←100
      rates←.03 .04  ⍝ rates of 3% then 4%
      ×/investment,1+rates
107.12
```

If you want to track the value of your investments over time, you can use the *scan* operator `\`.

```
      fv←{⍺××\1+⍵}
      rates←0 .03 .04 .06 .02 .02 .04 ⍝ the leading 0 is to show the initial amount
      2⍕100 fv rates                  ⍝ 2⍕ formats to 2 decimal places
100.00 103.00 107.12 113.55 115.82 118.13 122.86
```

Over time, you may want to invest or withdraw funds. For example:

```
      amounts←100 0 20 0 ¯10 0 0
```

Positive amounts are investments and negative amounts are withdrawals.

To calculate each period, you use can use a recurrence relation:

```
      r[n]←amounts[n]+r[n-1]×1+rates[n] ⍝ for n=2 through ≢amounts, r[1]←amounts[1]
```

This recurrence relation is easy to write as loop.

Task 1: Write a function named **`rr`** which has the following syntax:

```
      r ← amounts rr rates
```

where:

- the right argument, **`rates`**, is a numeric vector of interest rates where the first value is 0.
- the left argument, **`amounts`**, is numeric vector of deposit and withdrawal amounts, the first value is the initial deposit.
- the result, **`r`**, is a numeric vector of the cumulative values.

Notes:

- Although **`rr`** can be implemented using a looping solution, non-looping solutions will be given higher credit.

Example:

```
      2⍕ 100 0 20 0 ¯10 0 0 rr 0 .03 .04 .06 .02 .02 .04
100.00 103.00 127.12 134.75 127.44 129.99 135.19
```

Task 2 Introduction

A frequently encountered business decision is how to choose between alternative development projects. A project will typically have start-up costs (for example, construction and equipment), ongoing costs (for example, labor, materials and utilities), and, hopefully, revenue. A commonly-used technique to evaluate a project is to measure the present value of its expected cashflow. Present value is the concept that an amount of money today is worth more than that same amount of money in the future due to inflation.

To calculate present value, each future amount in the cashflow needs to be discounted by its corresponding cumulative interest rate and then the discounted values summed.

Task 2: Write a function named **pv** which will calculate the present value of a cashflow and has the following syntax:

```
      r ← cashFlow pv rates
```

where:

- the right argument, **rates**, is a numeric vector of interest rates where the first value is 0
- the left argument, **cashFlow**, is numeric vector representing a cash flow.
- the result, **r**, is a single number representing the present value of the cash flow.

Notes:

- While **pv** can be implemented using a looping solution, non-looping solutions will be given higher credit.

Examples:

```
      ¯6200 ¯2000 3400 3850 4300 4750 pv 0 .03 .04 .06 .02 .02
6156.480816

      ¯800 ¯3000 ¯2000 3400 3850 4300 pv 0 .03 .04 .06 .02 .02
4378.758757
```

Problem 6 – Merge (1 task)
Level of Difficulty: Medium

A common task is merging some sort of data into a template or electronic form. In this problem we'll be merging data stored in JSON files into a character vector template stored in a separate text file.

Notes:

- The merge areas of the template are delimited by @.
- Any @ that is not a part of a merge area (for example, it could be a part of an email address such as 'contest@dyalog.com') is doubled '@@' in the template.
- Any merge area that does not have a corresponding named variable in the namespace should be replaced with '???'.
- There is a sample template and two JSON files provided in the **Data** folder of the **Contest2020.zip** file - **template.txt**, **merge1.json** and **merge2.json**
  These files can be read using the `⎕NGET` system function:
  ```
  data←⊃⎕NGET filename
  ```
- The JSON data can be converted into an APL namespace using the `⎕JSON` system function:
  ```
  ns←⎕JSON data
  ```
- `ns.⎕NL ¯2` can be used to produce the namelist of variables in namespace `ns`.
- `ns⍎variable` can be used to retrieve the value of `variable` in `ns`.

Task 1: Write a function named **`Merge`** which has the following syntax:

```
      text ← templateFile Merge jsonFile
```

where:

- the right argument **`jsonFile`** is a character vector representing the name of a JSON file containing data to be merged.
- the left argument **`templateFile`** is a character vector representing the name of a text file containing the template to be merged into.
- the result, **`text`**, is a character vector representing the merged text according to the guidelines above.

Example:

```
      ⊢template←⊃⎕NGET '/Data/template.txt'
@salutation@ @firstname@ @lastname@;
Congratulations!  You have won a @prize@ worth over £@value@!
@firstname@, please come to our office to pick up your @prize@.
Please feel free to contact us at info@@contest.com.
Your email address in our domain is @firstname@@@contest.com

      ⊢merge1←⊃⎕NGET '/Data/merge1.json'
{
    "firstname":"Drake",
    "lastname":"Mallard",
    "prize":"yoyo",
    "value":100
}
      ns←⎕JSON merge1  ⍝ convert JSON to namespace
```

```
      ns.⎕NL ¯2        ⍝ variables in the namespace
┌─────────┬─────┬────────┬─────┐
│firstname│prize│lastname│value│
└─────────┴─────┴────────┴─────┘
      ns⍎'firstname'   ⍝ retrieve the value of firstname in the namespace
Drake

      '/Data/template.txt' Merge '/Data/merge1.json'
??? Drake Mallard;
Congratulations!  You have won a yoyo worth over £100!
Drake, please come to our office to pick up your yoyo.
Please feel free to contact us at info@contest.com.
Your email address in our domain is Drake@contest.com.
```

Problem 7 – UPC (3 tasks)
Level of Difficulty: Medium

The Universal Product Code (UPC) is a barcode that is widely used for tracking items in stores. The first known UPC-labeled item to be scanned was a 10-pack of Wrigley's Juicy Fruit chewing gum on 26 June 1974. That pack of gum is now on display at the Smithsonian American History Museum in Washington, DC.

Each UPC[1] barcode consists of a scannable strip of black bars and white spaces and a sequence of 12 digits. There is only one way to represent a 12-digit number visually and only one way to represent the black bars and white spaces numerically. Each digit is represented by a unique pattern of 2 bars and 2 spaces. The bars and spaces vary from 1-4 "modules" wide. The total width for a digit is always 7 modules.

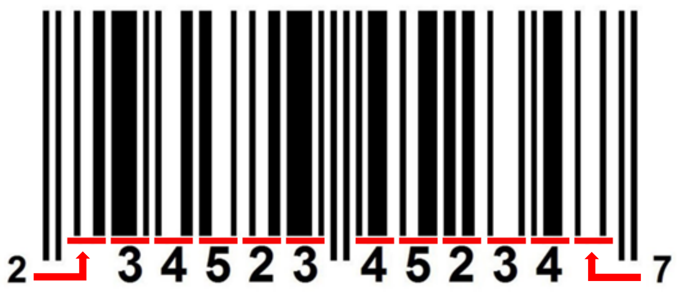
In the sample below, the annotations in red were added to show the delineation between digits and are not part of the barcode image itself. The scannable area of a UPC barcode follows the pattern `BSLLLLLMRRRRRCE`, where:

- `B` `M` and `E` are the beginning, middle, and end guard patterns. They are the vertically longer bars in the barcode and do not represent digits. B and E have the pattern bar-space-bar (101) and M has the pattern space-bar-space-bar-space (01010).
- `S` is the number system digit (the leftmost 2 in the sample). The number system digit is a convention that indicates the use of the barcode. For instance, 2 is generally used for items sold by variable weight (meats, fruits, vegetables, etc.), 3 indicates a National Drug Code (NDC) number in the U.S., 5 is used for coupons and so on.
- `LLLLL` are the 5 left side digits (34523 in the sample)
- `RRRRR` are the 5 right side digits (45234 in the sample)
- `C` is the check digit (7 in the sample) and can be used to as one verification that the code was scanned correctly. The check digit x_{12} satisfies the equation:

$$(3x_1 + x_2 + 3x_3 + x_4 + 3x_5 + x_6 + 3x_7 + x_8 + 3x_9 + x_{10} + 3x_{11} + x_{12}) \equiv 0\ (mod\ 10)$$

Or in APL:

```
0 = 10|x+.×12⍴3 1 ⍝ where x is the vector of digits
```



[1] There are several variations of UPC barcodes. For the purposes of this problem, the term "UPC" refers to the UPC-A variation.

Using a 0 (for space) or 1 (for bar) for each module, the digits 0-9 can be represented as follows:

Digit	Left Representation	Right Representation
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

Notice that the left representations have odd parity (the number of 1s is an odd number), whereas the right representations have even parity. The parity can be used to determine the direction in which the barcode was scanned. Notice also that the left and right representations are the Boolean negation of each other.

A complete UPC barcode consists of 95 modules – 12 digits of 7 modules each (84) + beginning (3), middle (5) and ending (3) guard patterns.

Task 1: Write a function named `CheckDigit` which has the following syntax:

```
      digit ← CheckDigit digits
```

where:

- the right argument **`digits`** is an 11-element integer vector representing the first 11 digits of the UPC code.
- the result **`digit`** is the integer check digit.

Examples:

```
      CheckDigit 2 3 4 5 2 3 4 5 2 3 4 ⍝ from the sample barcode above
7
      CheckDigit 0 4 3 3 9 6 5 4 6 7 9 ⍝ Charlie's Angels 4K Ultra HD disc
0
      CheckDigit 3 8 1 3 7 0 0 3 8 4 4 ⍝ Aveeno Daily Moisturizing Lotion
3
```

Task 2: Write a function named `WriteUPC` which has the following syntax:

```
      bits ← WriteUPC digits
```

where:

- the right argument **`digits`** is an integer vector of 11 elements representing the digits (in left to right order) to be represented in the barcode.
- the result, **`bits`**, is a 95-element Boolean vector representing the modules of the UPC barcode in left to right order. If there is an error in **`digits`** like incorrect length, or element(s) not in 0-9, return ¯1.

Your function needs to calculate the check digit to include as the 12th digit of the barcode; you may use `CheckDigit` from the previous task to do this.

Examples: Your function should return a Boolean vector. The examples use that Boolean to index into '01' so that the results can be displayed on a single line (albeit in a smaller font).

```
      '01'[1+WriteUPC 2 3 4 5 2 3 4 5 2 3 4]
10100100110111101010001101100010010011011110101010101110010011101101100100001010111001000100101

      WriteUPC 2 3 4 5 2 3 4 5 2 3 ⍝ too few digits
¯1

      WriteUPC 11⍴42 ⍝ numbers not ∊ 0-9
¯1
```

Task 3: Write a function named `ReadUPC` which has the following syntax:

```
      digits ← ReadUPC bits
```

where:

- the right argument **`bits`** is a Boolean vector representing the scanned bits of the UPC barcode. **`bits`** could be the result of scanning from right to left or left to right.
- the result, **`digits`**, is an integer vector of the digits of the UPC barcode in left to right order. If there is an error in the barcode, like incorrect parity, incorrect number of bits, or the check digit is not correct, return ¯1.

Examples:

```
      ReadUPC WriteUPC 2 3 4 5 2 3 4 5 2 3 4
2 3 4 5 2 3 4 5 2 3 4 7

      ReadUPC ⌽WriteUPC 2 3 4 5 2 3 4 5 2 3 4
2 3 4 5 2 3 4 5 2 3 4 7

      ReadUPC 1,WriteUPC 2 3 4 5 2 3 4 5 2 3 4 ⍝ too many bits
¯1
      ReadUPC 1 0 0 1 0 0 0@(85+⍳7)⊢WriteUPC 2 3 4 5 2 3 4 5 2 3 4 ⍝ bad check digit (8 not 7)
¯1
```

Problem 8 – Balancing the Scales (1 task)
Level of Difficulty: Hard

Task 1: Write a function named **`Balance`** which has the following syntax:

```
        parts ← Balance nums
```

where:

- the right argument, **`nums`**, is an integer vector of 2 to 20 elements
- the result, **`parts`**, is a 2-element vector of integer vectors where the sums of the elements are equal, and the concatenation of **`parts`** has the same elements as in **`nums`**. In other words, if possible, parts should satisfy the following:

  ```
      =/+/¨parts               ⍝ both parts have the same total
      ≡/{ω[⍋ω]}¨nums (∊parts)  ⍝ (∊parts) and nums have the same elements
  ```

  If `nums` cannot be split into 2 equally summed groups, return **`⍬`**.

Notes:

- There may be more than correct result. Your solution is correct as long as its result meets the above criteria.
- Understanding the nuances of the problem is the key to developing a good algorithm.

Examples: (using `]boxing on`)

```
      Balance 1 2 3 4 5 ⍝ odd total?  Can't split evenly, so return ⍬

      Balance 1 3 ⍝ can't split evenly, so return ⍬

      Balance 10 81 98 27 28 5 1 46 63 99 25 39 84 87 76 85 78 64 41 93
┌────────────────────┬──────────────────────────────────────┐
│81 98 46 63 99 85 93│10 27 28 5 1 25 39 84 87 76 78 64 41│
└────────────────────┴──────────────────────────────────────┘

      Balance 90 44 76 48 41 50 9 69 30 74 15 56 28 31 52 97 81 78 22 34 ⍝ returns ⍬

      Balance 1 1 1 1 1 1 1 1 1 1 1 1 1 13
┌──┬─────────────────────────┐
│13│1 1 1 1 1 1 1 1 1 1 1 1 1│
└──┴─────────────────────────┘
      Balance ⍳8
┌─────┬─────────┐
│3 7 8│1 2 4 5 6│
└─────┴─────────┘
      Balance ⍳11
┌─────────┬─────────────┐
│3 9 10 11│1 2 4 5 6 7 8│
└─────────┴─────────────┘
```

Problem 9 – Upwardly Mobile (1 task)
Level of Difficulty: Hard

Imagine you work for IBM (no, not *that* IBM, you work for Incredibly Big Mobiles) as a production planner. The mobile designers send you files representing their latest mobile designs. It's your job to calculate the weights that will keep the mobiles in balance so that you can ensure there's sufficient inventory and that the production line knows how to construct the mobiles.

In this problem you're going to write a program that will:

- Read a file containing a diagram which represents a mobile.
- Parse the file to determine the coefficients for each weight.
- Solve the matrix of coefficients and scale them to return a vector representing the set of smallest integer weights that will keep the mobile in balance.

Notes:

- We recommend you use `⎕NGET` to read the file.
- Each file will have content that will look something like this:

```
    ┌───────┴───────────────┐
┌───┴───────┐           ┌───┴───┐
A           B           │       C
                     ┌──┴──┐
                     D     E
```

- The ┴ character represents a balance point (fulcrum).
- The letters represent the weights and will always appear in alphabetic order from top to bottom, left to right.
- The ratio of weights on each side of the balance point is determined by the number of characters to (and including) the endpoint (either ┌ or ┐). In the diagram above, the ratio A:B is 4:8, and D:E is 3:3.
- The diagram above represents the following relationships:

```
     4A  =  8B
     3D  =  3E
 4(D+E)  =  4C
 8(A+B)  =  16(C+D+E)
```

- Eventually you'll construct a matrix of coefficients and use the *matrix divide* function `⌹` to solve for a set of weights.
- There are 3 files provided in the **Data** folder of the **Contest2020.zip** file - **mobile1.txt**, **mobile2.txt**, and **mobile3.txt**.

Task 1: Write a function named `Weights` which has the following syntax:

```
      weights ← Weights filename
```

where:

- the right argument **`filename`** is a character vector representing the name of a file.
- the result **`weights`** is a vector representing the set of smallest integers that will keep the mobile in balance. **`weights`** should be in "alphabetic" order according to the diagram.

Examples: (these results are correct for the supplied files)

```
      Weights¨'/Data/mobile1.txt' '/Data/mobile2.txt' '/Data/mobile3.txt'
┌───┬────────────┬─────────────────┐
│2 1│16 8 6 3 3│21 14 8 13 16 12│
└───┴────────────┴─────────────────┘
```

Judges' comment on this problem:
This is the most difficult problem in the competition and likely requires an iterative or recursive solution. (If you find an elegant, array-oriented solution, we'll be really impressed!)