How I Won the APL Problem Solving Contest

TzuChing Lee

12 Oct. 2022

Who am I

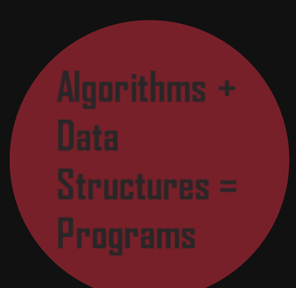
- Lee Tzu-Ching 李子敬
- From Taiwan
- Master student @ NTHU
  - Studying quantum algorithm
- Interested in
  - array languages, combinator, FP ...
  - geography, drawing, math, anime stuff

How I found APL

– 2020 ~ 2021 –



– 2021 ~ 2022 –




Why I like APL

Glyphs

⍤⍥⍨⍣⍋⊆≠≤

Concise

```
std::reduce(v.cbegin(), v.cend(),
       0u, std::plus<>{});
```

+⌿

Algo. as primitive

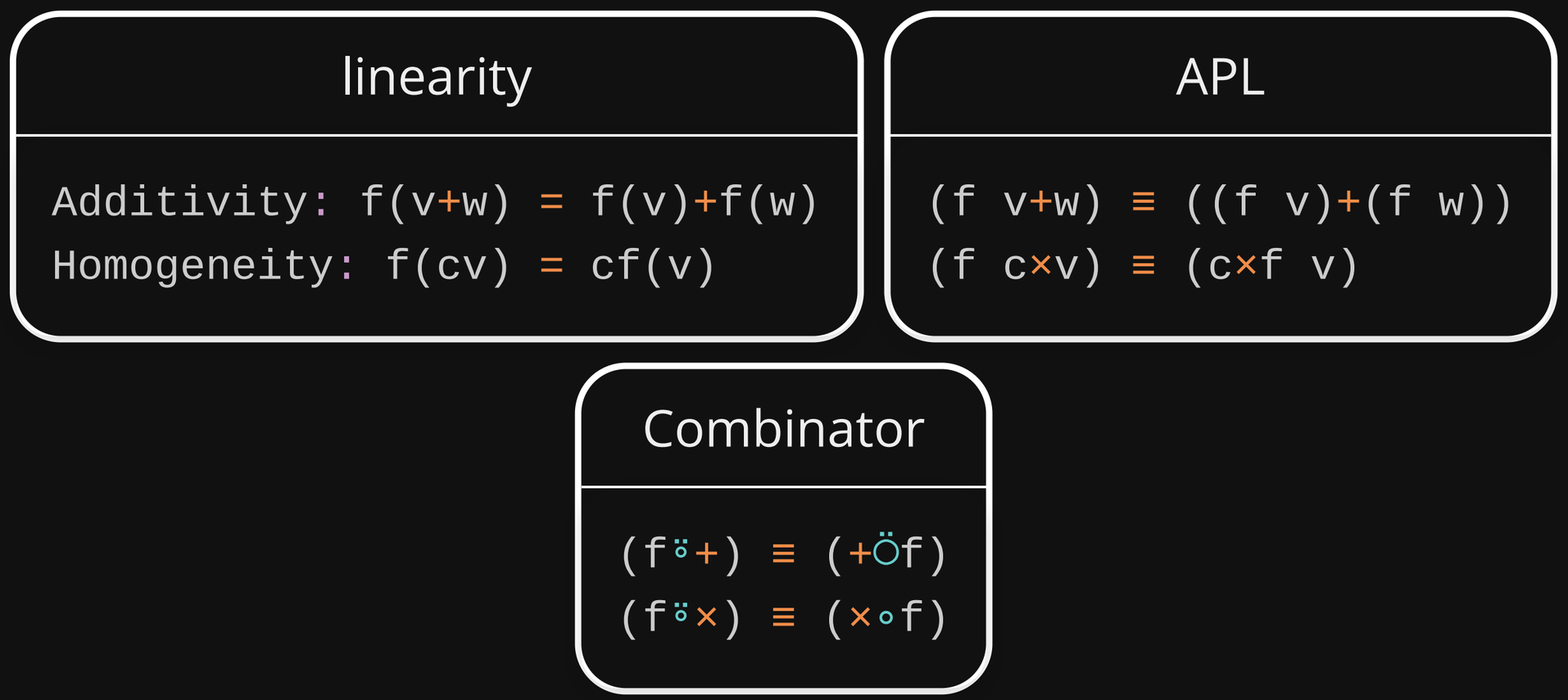
⍒+-⍋/ ← → ⊥⌊⍒⍋u

Operators

⌿@⍣⌸⍤

Combinator + Trains

(fgh)∘(fg)⍤(A⍨fg)⍥f


linearity
Additivity: f(v+w) = f(v)+f(w)
Homogeneity: f(cv) = cf(v)
APL
(f v+w) ≡ ((f v)+(f w))
(f c×v) ≡ (c×f v)
Combinator
(f⍤+) ≡ (+⍤f)
(f⍤×) ≡ (×∘f)

The Competition

1. `runs` ◇ `fill` ◇ `subspace`
2. `reshape` ◇ `reshape2`
3. `Attended` ◇ `ShowedUp` ◇ `Popular`
4. `Ballot` ◇ `IRV`
5. `Base85`
6. `DDN`

My solusions : fars.ee/jvax

Problem 2

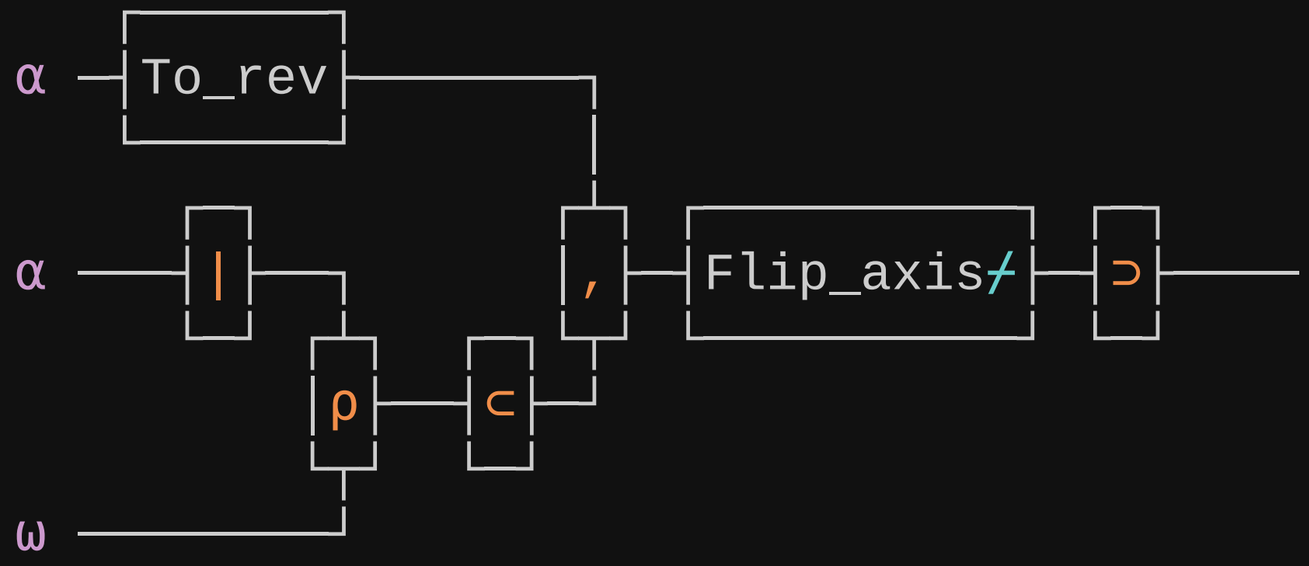
– Reshaping Reshape –

reshape

Extend `⍴` so the left argument can contain negative integers. And the output is reversed along those axis.

```
    5 reshape ⍳3
1 2 3 1 2
    ¯5 reshape ⍳3
2 1 3 2 1
    ¯2 ¯5 reshape ⎕A
J I H G F
E D C B A
     3 ¯3 reshape ' '(≠⊆⊢)'columns are reversed'
┌─────────┬───┬───────┐
│reversed│are│columns│
├─────────┼───┼───────┤
│reversed│are│columns│
├─────────┼───┼───────┤
│reversed│are│columns│
└─────────┴───┴───────┘
```

```
reshape ← {
  To_rev ← ⍳0>,
  Rho ← |⍤⊣ ⍴ ⊢
  Flip_axis ← {⊖[⍺]⍵}
  ⊃ ⍺ Flip_axis⌿⍤(To_rev⍤⊣ , ⊂⍤Rho) ⍵
}
```



reshape2

> *Extend* `reshape` *so the left argument can contain up to one special value in* `{±0.5, ±1.5, ±2.5}`.

When the data (right argument) doesn't fill the corresponding axis, it preprocesses the data before passing it to `reshape` :

- `±0.5` : Truncate the data.
- `±1.5` : Repeat the data.
- `±2.5` : Pad the data, like what `X↑Y` does.

```
     0.5 6 reshape2 ⍳7 ⍝ 0.5 truncate the data
1 2 3 4 5 6
     1.5 6 reshape2 ⍳7 ⍝ 1.5 repeat the data
1 2 3 4 5 6
7 1 2 3 4 5
     2.5 6 reshape2 ⍳7 ⍝ 2.5 pad the data
1 2 3 4 5 6
7 0 0 0 0 0
    ¯2.5 3 reshape2  'reverse' 'as' 'well' 'it' 'handles'
```

it	handles	
reverse	as	well

```
reshape2 ← {
  dim ← ,⍺
  spd ← dim[spid ← ⍸0≠1|dim]
  unit ← |spd÷⍤≈ (⊢ ×.* 0∘≠) dim
  ⍝ Expszif ← (≠,⍵) (⊣+(⊢|-⍤⊣)) unit*⊢
  Expszif ← (-≠,⍵)(|⍤≈-⊣)unit*⊢
  dim[spid] ← (×spd)×⌊unit÷⍤≈Expszif 0.5≠|spd
  expsz ← Expszif 2.5=|spd
  dim reshape expsz↑,⍵
}
```

Problem 5

– Base85 –

Base85

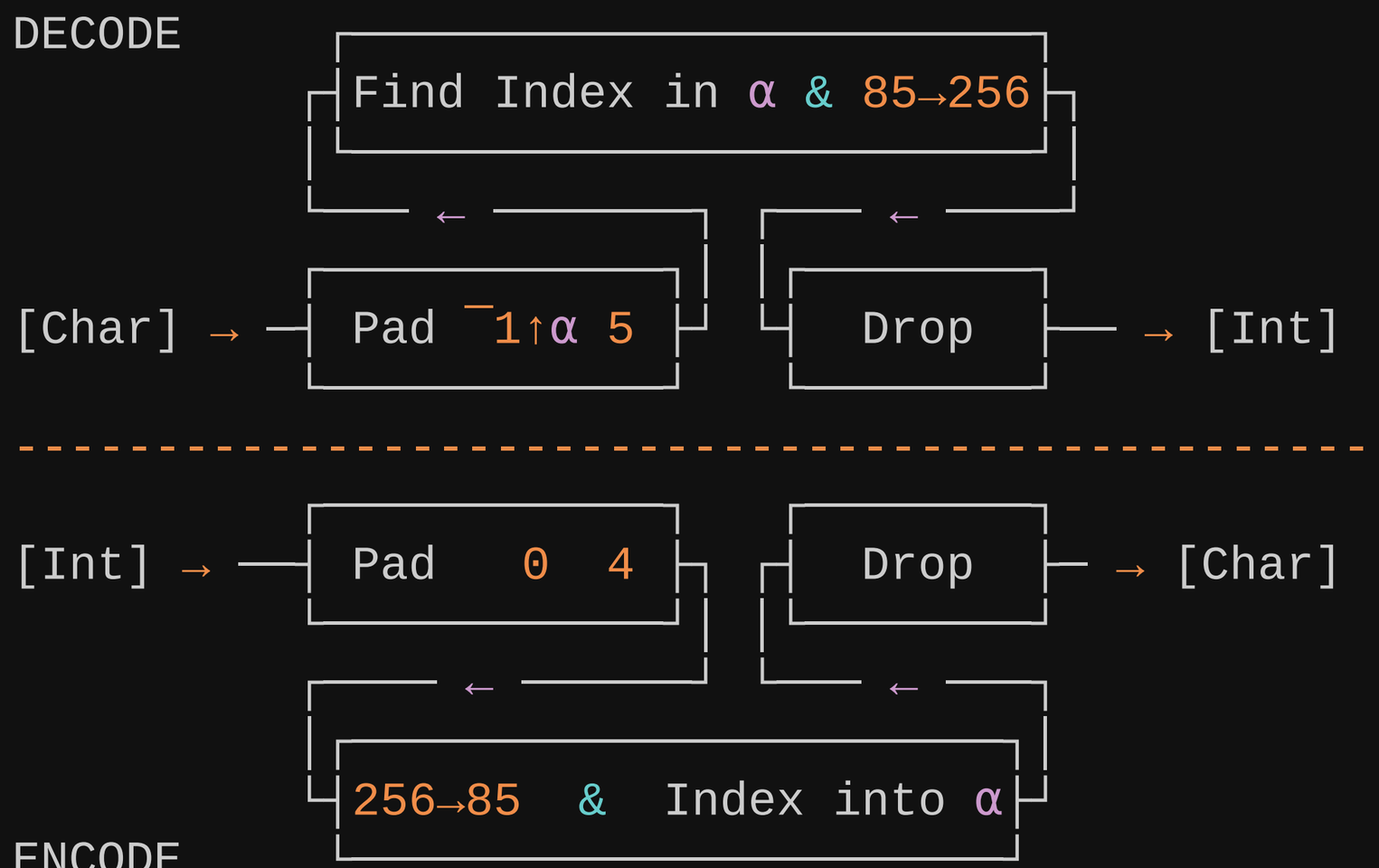
> ***Base85*** *is a binary-to-text encoding. It uses* ***five*** *ASCII characters to represent* ***four*** *bytes of binary data.*
>
> *It's more effecient than* ***Base64****, which uses four characters to encode three bytes.*

The task is to write a dyadic function `Base85` that takes

1. A character vector ⍺ (with length 85).
2. A vector ⍵.
   - numeric ϵ [0,255]* : encode ⍵ to a character vector.
   - character : decode ⍵⍳⍺ to a numeric vector.

```
      O ← ⎕UCS 32+⍳85
      Q ← 'ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖØÙÚÛÜÝßàáâãäåæçèéêë'
      Q,← 'ìíîïðñòóôõöøùúûüþΔⒶⒷⒸⒹⒺⒻⒼⒽⒾⓀⓁⓂⓃⓄⓅⓆⓇⓈⓉⓊⓋⓌⓍⓎⓏ'
      O Base85 ⎕UCS 'Dyalog 22' ⍝ Encoding
7!W3WDeC1J1&
      ⎕UCS O Base85 '7!W3WDeC1J1&' ⍝ Decoding
Dyalog 22
      ⎕UCS O Base85 'vwv非7!W法3WDe字C1 {元}~{xyz} J1&⍒⍋'
Dyalog 22
      Q (Base85⍣2 ≡ ⊢) ⎕UCS 'It should be an involution'
1
      text ← '{7}ØⓅåⒶðÚ{!W3WD}êÖ{eC}ÏÔÓÄø{1J1&}ⓈÎÊΔÛ'
      ⎕UCS O Base85 text ⍝ Decode with different char. set
Dyalog 22
      Q⍞⍨⊂ ⎕IO+ Q Base85 text
ⓅⓄⓇⓉⓊⒼⒶⓁΔⓄⓁⒽãⓄ
```

How Base85 Works



```
_Flip ← {⍉⍺⍺⍉⍵}
_Stack_ ← {∈ ⍺⍺ ((⍵⍵÷⍨≠⍵) ⍵⍵)⍴⍵}
_Padto_ ← {(-⍴)↓ ⍺⍺ ⍵,⍺⍴⍨⍴←⍵⍵|-≠⍵}
_U_Reshape_ ← {
    f w ← ⍵⍵
    f ((⍺⍺ _Flip _Stack_ w) _Padto_ w) ⍵
}
Base85 ← {
     Tsf ← (4⍴256)⊤(5⍴85)⊥1-⍨⍺∘⍳
     Dec ← (Tsf⍣¯1) _U_Reshape_ ((¯1↑⍺) 5) ∩∘⍺
     Enc ← (Tsf⍣¯1) _U_Reshape_ (0      4) ⊢
     2|⎕DR ⍵: Enc ⍵ ⋄ Dec ⍵
}
```

```
_U_Reshape_ ← {
    f w ← ⍵⍵
    f ((⍺⍺ _Flip _Stack_ w) _Padto_ w) ⍵
}
⎕IO ← 0
Base85 ← {
     a ← 1⌽⍳256
     Tsf ← (4⍴256)⊤(5⍴85)⊥⍺∘⍳
     Dec ← (Tsf⍣¯1) _U_Reshape_ ((¯1↑⍺) 5) ∩∘⍺
     Enc ← (Tsf⍣¯1) _U_Reshape_ ((¯1↑a) 4) ∩∘a
     2|⎕DR ⍵: Enc ⍵ ⋄ Dec ⍵
}
```

```
⎕IO ← 0
_U_Fmt_ ← {
    q r ← ⌈⍵⍵(÷⍨,|∘-)≠d ← ⍵∩⍺
    (-r)↓∈⍉ ⍺⍺ ⍉(q ⍵⍵)⍴d,r⍴¯1↑⍺
}
Base85 ← {
    a ← 1⌽⍳256
    Tsf ← (4⍴256)⊤(5⍴85)⊥⍺∘⍳
    Dec ← (Tsf⍣¯1) _U_Fmt_ 5
    Enc ← (Tsf⍣¯1) _U_Fmt_ 4
    2|⎕DR ⍵: a Enc ⍵ ⋄ ⍺ Dec ⍵
}
```

```
┌──────────────┐
│ Pad & Drop   │
│  & filter    │
└──────────────┘
```

```
_CharEncByte_ ← {
    a ← 1⌽⍳256
    Tsf ← (⍵⍵⍴≠a)T(ααρ≠α)⊥α∘⍳
    Dec ← (Tsf⍤⍤ 1) _U_Fmt_ αα
    Enc ← (Tsf⍤⍤¯1) _U_Fmt_ ⍵⍵
    2|□DR ⍵: a Enc ⍵ ⋄ α Dec ⍵
}
Base85 ← (5 _CharEncByte_ 4)
```

Base85 *is a binary-to-text encoding. It uses* ***five*** *ASCII characters to represent* ***four*** *bytes of binary data.*

It's more effecient than ***Base64****, which uses four characters to encode three bytes.*

```
Base64 ← {
    c ← ⎕A,(⎕C⎕A),⎕D,'+/' ⍝ A-Za-z0-9+/
    r ← c (4 _CharEncByte_ 3) ⍵
    2|⎕DR r : r ⋄ r,(4|-≢r)⍴'='
}
```

Advice

1. Do experiments.
2. Try to achieve different goals.
3. Comment your code.
4. Rewrite the solution.
5. Have fun.

Where & How

- Videos :
  - Conor, Adám, John, Rodrigo, Dyalog ...
- Podcasts :
  - ADSP & The Array Cast
- Websites :
  - dfns, J Wiki, APL Wiki, BQN tutorials, stackoverflow
- Tutorials :
  - learnapl, mastering Dyalog APL (Book)
- REPL :
  - RIDE, tryapl.org, APL Cart

Suggestions

- Make sure the problem description gives all information
  - In Base85, filtering was only mentioned in example
- Naming style
- Show scores and the weighting of them after the contest (?
- `f≍∘g≍`

Plans

- Know more about 
- Learn how APL interacts with the system
- Try out array languages like BQN, K, ...
- Do more projects in array languages
- Theoretical aspect of array languages (?

Question?

Problem 1

– Sub-space Journey –

fill

- `α`: non-negative numeric vector
- `ω`: a `?-by-(2×≢α)` matrix whose rows are (start, shape) pairs
- return: an integer array of shape `α`, where each sub-space is filled by the corresponding row index in `ω`, other positions sould be 0.

```
     3 3 fill 1 4 ρ 2 2 2 2
0 0 0
0 1 1
0 1 1
     3 3 fill 2 4 ρ 2 2 2 2  1 1 1 3
2 2 2 ⍝ Numbered by row index
0 1 1
0 1 1
     3 3 fill 3 4 ρ 2 2 2 2  1 1 1 3  2 1 2 2
2 2 2 ⍝ Prefer larger number when overlapping
3 3 1
3 3 1
     12 fill 4 2 ρ 1 1  3 4  4 0  5 6
1 0 2 2 4 4 4 4 4 4 0 0      ⍝ Works in 1 dimension
```

```
fill←{
    rank←≢⍺
    Format←{
        mark←1 0⍴⍨2×rank
        Shuffle←(⊂⍋⍒mark)⌷⍵
        Group←mark⌿⊂⍤1
        ⍺Group⍺Shuffle⍺⍵
    }
    Run←{⍺↑1↓(/⍤0 1⍤1)⍵}
    Layer←(⍤.∧).Run
    Combine←⊃⌈.×⍤(⍳⍤≢)⍨
    ⍺ Combine⍤Layer∘Format ⍵
}
```

```
fill ← {
  Pat ← 1 0 ρ⍨ ≢∘⍉
  Shf ← (⊂∘⍋ ⍒⍤⊣) ⌷[2] ⊢
  Grp ← ⊂⍤1
  Run ← {α↑1↓(/∘0 1⍤1)ω}
  ⍝ | A | B | -> |A1 B1|A2 B2| ...
  Arrange ← ⍉ Pat (⊣ Grp Shf) ⊢
  BuildUp ← (∘.∧).Run
  Combine ← ⊃ ⌈.×∘(⍳∘≢)⍨
  α Combine⍤BuildUp∘Arrange ω
}
```