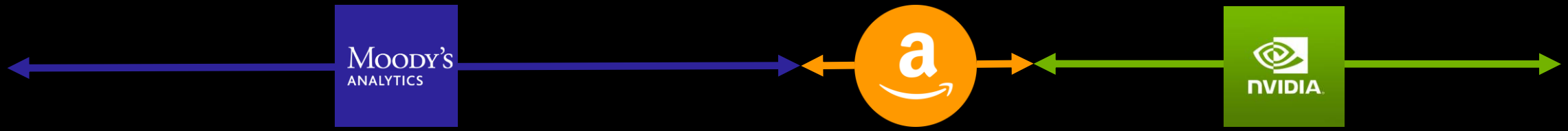
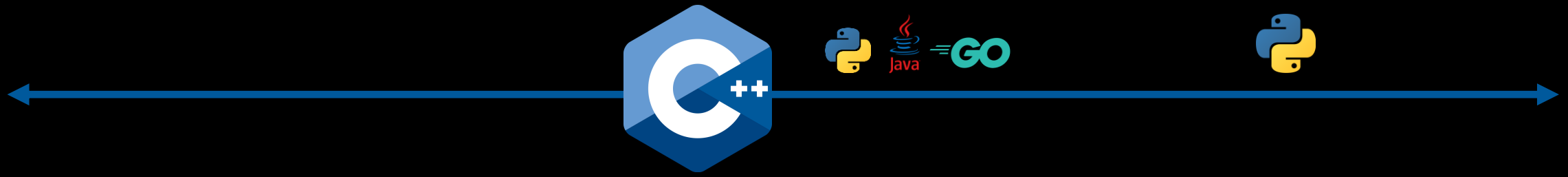
the Twin Algorithms

Conor Hoekstra

 code_report 





About Me

Conor Hoekstra / @code_report

THE STL ALGORITHM CHEAT SHEET

by @code_report

ZIP ALGORITHMS

inner_product	zip_reduce
transform_reduce[17]	zip_reduce
transform	zip_with
mismatch	zip_find_not
equal	zip_reduce*

ORDER LOGN ALGORITHMS

- binary_search
- lower_bound
- upper_bound
- equal_range
- partition_point

CODE REVIEW A

sort	O(nlogn)
partial_sort	O(n) - $O(n^2)$
nth_element	O(n)

CODE REVIEW B

find_if	O(n)
lower_bound	O(logn)

ALGORITHM RELATIONSHIPS

is_sorted -> is_sorted_until -> adjacent_find -> mismatch

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce[17]	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
partial_sum inclusive_scan[17]	1	Yes	Map	plus{}
find_if	1	No	Reduce	-
	find, all_of, any_of, none_of			
transform	1/2	No	Map	-
	replace[17], replace_if[17]			
adjacent_difference	2	No	Map	minus{}
inner_product transform_reduce[17]	1/2	Yes	Reduce	plus{} multiplies{}
transform_inclusive_scan[17]	1/2	Yes	Map	-
mismatch	1/2	No	Reduce	equal{}
adjacent_find	2	No	Reduce	equal{}

Note: non-accumulator reductions all short-circuit

THE TWIN ALGORITHMS

to be announced (at a future conference)

THE STL ALGORITHM CHEAT SHEET

by @code_report

ZIP ALGORITHMS

```
inner_product
transform_reduce[17]
transform
mismatch
equal
```

```
zip_reduce
zip_reduce
zip_with
zip_find_not
zip_reduce*
```

ORDER LOGN ALGORITHMS

```
binary_search
lower_bound
upper_bound
equal_range
partition_point
```

CODE REVIEW A

sort	O(nlogn)
partial_sort	O(n) - O(n^2)
nth_element	O(n)

CODE REVIEW B

find_if	O(n)
lower_bound	O(logn)

ALGORITHM RELATIONSHIPS

```
is_sorted -> is_sorted_until -> adjacent_find -> mismatch
```

THE ALGORITHM INTUITION TABLE

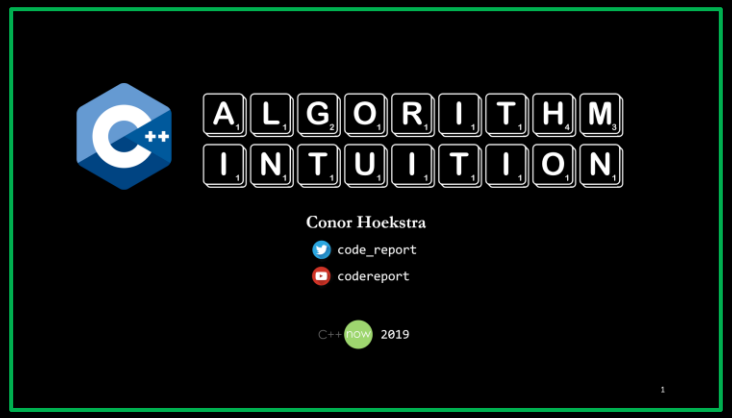
Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce[17]	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
partial_sum inclusive_scan[17]	1	Yes	Map	plus{}
find_if	1	No	Reduce	-
	find, all_of, any_of, none_of			
transform	1/2	No	Map	-
	replace[17], replace_if[17]			
adjacent_difference	2	No	Map	minus{}
inner_product transform_reduce[17]	1/2	Yes	Reduce	plus{} multiplies{}
transform_inclusive_scan[17]	1/2	Yes	Map	-
mismatch	1/2	No	Reduce	equal{}
adjacent_find	2	No	Reduce	equal{}

Note: non-accumulator reductions all short-circuit

THE TWIN ALGORITHMS

to be announced (at a future conference)

The Algorithm Intuition Trilogy

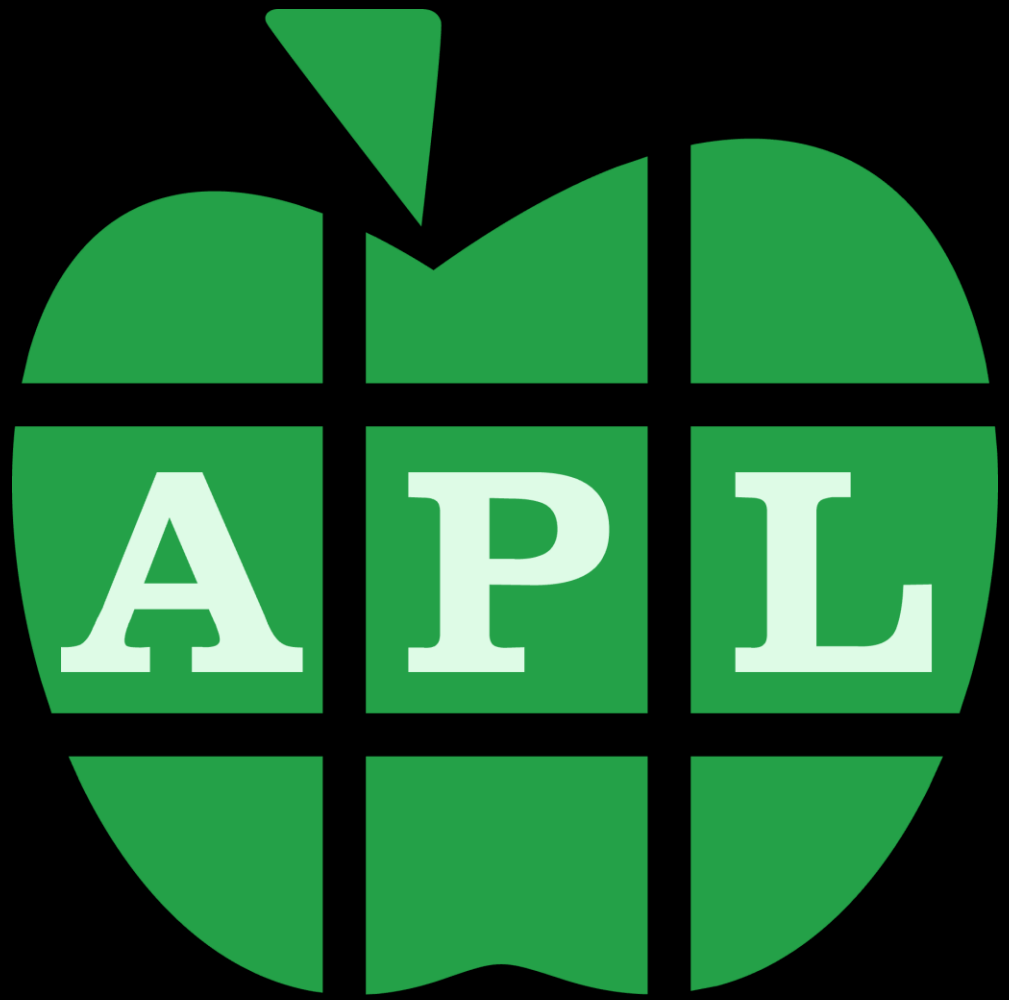



Conor Hoekstra
@code_report
Just submitted my @cppnow 2020 talk submission, a talk I am calling "The Twin Algorithms." Please C++ gods - let me give this talk - I am more excited about this talk than I have been about anything else in my career. #cpp #cplusplus
the
Twin Algorithms
Conor Hoekstra
code_report
NVIDIA. RAPIDS
#include
2:32 PM · Jan 7, 2020 · Twitter Web App
View Tweet analytics
3 Retweets 1 Quote Tweet 48 Likes

the Twin Algorithms

Conor Hoekstra

 code_report 


APL

"The power of notation cannot be easily understood unless you have something to provide context. I find that music is good for this. **Musical notation**, to a trained reader, communicates with great efficiency what a musician needs to know in order to reproduce a piece. It would be unimaginable to write music using words or ASCII symbols."

Algoshift

Comment on HackerNews post: What's cool about APL?

“A lot of the comments about APL (going back decades) go something like "it looks so strange" or "it's hard to read". The problem with this approach to the language is that it ignores that, in **APL**, notation has a purpose. And in order to understand this purpose and the advantages offered **you have to learn the notation**. An opinion does not count unless the person issuing that opinion has taken the time to study the language and gain some perspective. Going back to the musical example, it would be like a non-musician complaining that musical notation is "strange" and "hard to read". Well, yeah! It is! If you don't study it. And, without learning this notation you can't be in a position to appreciate what it does and how useful it is.”

Algoshift

Comment on HackerNews post: What’s cool about APL?

"Most university computer scientists don't really know **APL**. They haven't appreciated what it means to think in **APL**."

Alan J. Perlis

In Praise of APL: A Language for Lyrical Programming

"Would one accept the view of a lecturer, about a poem by Pushkin, that the **poetry** is bad; if they could not read **Russian**? Certainly not! It is the same if one asks programmers inexpert in **APL** to form a judgment concerning the readability of programs written in **APL**. Relying on their status as professionals, they assert that these programs are unreadable…
and people believe them!"

Bernard Legrand

APL – a Glimpse of Heaven

"Because of its cryptic appearance, it is almost impossible to convince anyone who might become interested in the beauty of **APL**, simply by showing them some subtleties and some attractive algorithms.

Do not try to convince anyone by showing that you can do with 10 symbols, what would take him 100 convoluted instructions: all the world prefers reading 100 lines of good (or even bad) English faced with 10 **Chinese** ideograms! You will only convince those who are willing to learn."

Bernard Legrand

APL – a Glimpse of Heaven

"Because of its cryptic appearance, it is almost impossible to convince anyone who might become interested in the beauty of **APL**, simply by showing them some subtleties and some attractive algorithms.

Do not try to convince anyone by showing that you can do with 10 symbols, what would take him 100 convoluted instructions: all the world prefers reading 100 lines of good (or even bad) English faced with 10 **Chinese** ideograms! **You will only convince those who are willing to learn**."

Bernard Legrand

APL – a Glimpse of Heaven

This talk **IS NOT** about
convincing you to learn **APL**.

What?

History

APL

C++11/14/17/20/23/2x

Alexander Stepanov

Sean Parent

Arthur Whitney

Bill Gates

Harry Potter

iota

100% Passion ❤

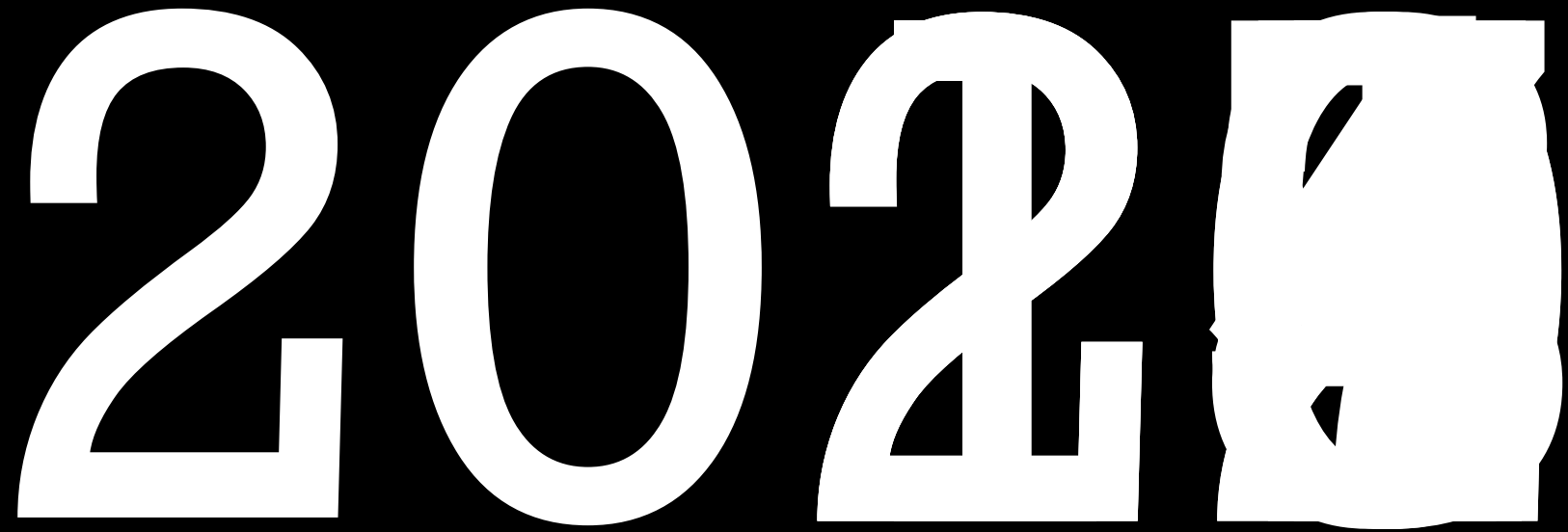
How?

How did a 

developer fall

in love with

?


2024

2010: ACMA 320

2010 SP

Course		Description	Attempted	Units/Grade		Points
ACMA	320	Actuarial Mathematics I	5.00	5.00	B	15.000
BUS	254	Managerial Accounting I	3.00	3.00	C	6.000
ECON	103	Principles of Microeconomics	4.00	4.00	A-	14.680
STAT	330	Math.Statistics	3.00	3.00	B	9.000
	TERM GPA : 2.980	TERM TOTALS :	15.00	15.00		44.680
	CUM GPA : 3.140	CUM TOTALS :	58.00	58.00		182.380

2010: ACMA 320

2014: MetSim

2017: KDB+ / Q

2018-05: iota

2019-01: #iotashaming

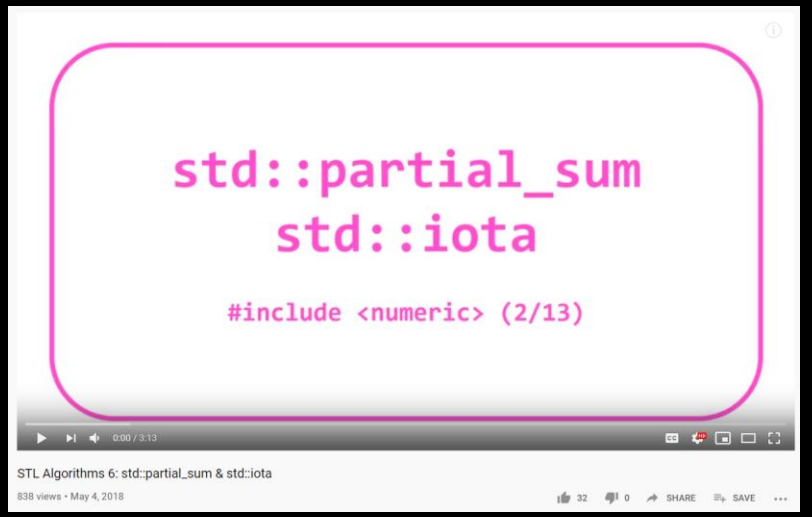
2019-06: Functional Geekery

Course		Description
ACMA	320	Actuarial Mathematics I

METSIM INTERNATIONAL

kx

λ Functional Geekery



2019-06: Functional Geekery

2018-08-08: Paul Leslie tweeted

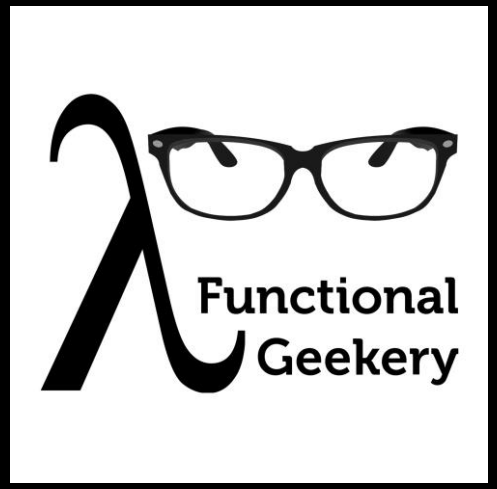
2018-08-09: CppCast Episode 162

2019-06: Functional Geekery

2018-08-08: Paul Leslie tweeted
2018-08-09: CppCast Episode 162
2018-08 to 10: Listen to all LambdaCasts
2018-10: Found D. Koontz on Fn Geekery Ep 105
2018/2019: Listen to all Functional Geekery
2019-06: Three episodes - 64, 65 & 76

Paul Leslie @michtiecpp · Aug 8, 2018

I feel like I am **cheating** on **@cppcast** with @lambdacast, but it is so good!

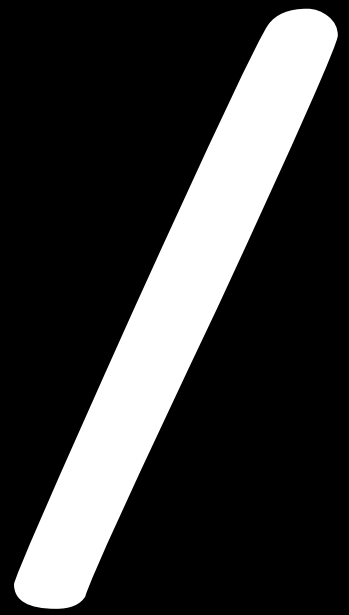


Episode 64: Alex Weiner

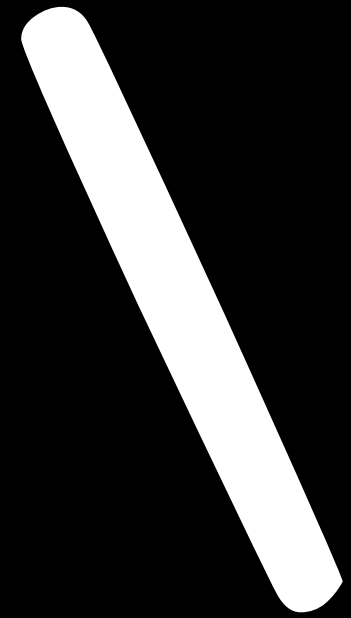
Episode 65: Morten Kromberg

Episode 76: Anthony Cipriano


```
      ⍳5
1 2 3 4 5
      +/⍳5
15
      +\⍳5
1 3 6 10 15
```



reduce



scan


```
      ⍳5
1 2 3 4 5
      +/⍳5
15
      +\⍳5
1 3 6 10 15
```


```
      ⍳5
1 2 3 4 5
      +/⍳5
15
      +\⍳5
1 3 6 10 15
```


```cpp
// iota 5
auto a = std::vector<int>(5);
std::iota(a.begin(), a.end(), 1);
// + reduce iota 5
auto t = std::vector<int>(5);
std::iota(t.begin(), t.end(), 1);
auto b = std::accumulate(t.cbegin(), t.cend(), 0);
// + scan iota 5
auto c = std::vector<int>(5);
std::iota(c.begin(), c.end(), 1);
std::partial_sum(c.cbegin(), c.cend(), c.begin());

for (auto e : a) std::cout << e << ' ';
std::cout << '\n';
std::cout << b << '\n';
for (auto e : c) std::cout << e << ' ';
std::cout << '\n';
```

https://godbolt.org/z/jxa4jcPYh


```
⍳5+/⍳5+\⍳5
```

```
auto a = s
```


```
+/⍳5
+\⍳5
```


```
+/⍳5   accumulate
+\⍳5   partial_sum
```

C++ now 2019 MAY 6-10 cppnow.org

Conor Hoekstra

Algorithm Intuition

Video Sponsorship Provided By: JET BRAINS

```cpp
template<class T>
using rev = reverse_iterator<T>;

int trap(vector<int>& v) {
    vector u(v.size(), 0);
    auto it = max_element(begin(v), end(v));
    inclusive_scan(begin(v),  next(it), begin(u),  ufo::max{});
    inclusive_scan(rbegin(v), rev(it),  rbegin(u), ufo::max{});
    return transform_reduce(cbegin(u), cend(u), cbegin(v), 0,
        std::plus<>(),
        std::minus<>());
}
```

146

https://www.youtube.com/watch?v=48gV1SNm3WA

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce[17]	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
adjacent_reduce	2	Yes	Reduce	-
partial_sum inclusive_scan[17]	1	Yes	Map	plus{}
adjacent_inclusive_scan	2	Yes	Map	-
find_if	1/2	No	Reduce	-
	find, all_of, any_of, none_of			
adjacent_find_if	2	No	Reduce	equal{}
transform	1/2	No	Map	-
	replace[17], replace_if[17]			
adjacent_transform	2	No	Map	-
inner_product transform_reduce[17]	1/2	Yes	Reduce	plus{} multiplies{}
transform_inclusive_scan[17]	1/2	Yes	Map	-

Note: non-accumulator reductions all short-circuit


```
+/⍳5   accumulate
+\⍳5   partial_sum
```

reduce
reductions

fold
cumulativeFold

fold | reduce
running(fold | reduce)

1. The brevity
2. Realization of /\ relationship


APL


APL

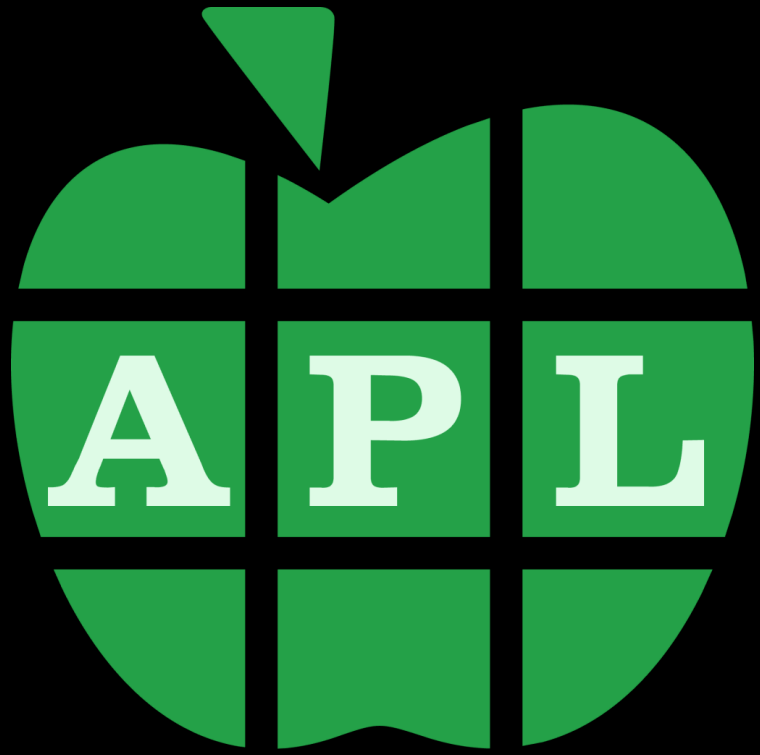
1. The brevity
2. Realization of /\ relationship
3. There is a ϕ primitive

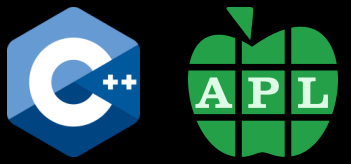
Overview

1. ☑ Introduction
2. ☑ How?
3. ☑ What?
4. ☑ Why?
5. ☐ Connections
6. ☐ History
7. ☐ `reduce`
8. ☐ `scan (filter_html_tags)`
9. ☐ `outer_product (count_lrud)`
10. ☐ `rotate`

Connections

Connections between



Connections between 

STL and Its Design Principles

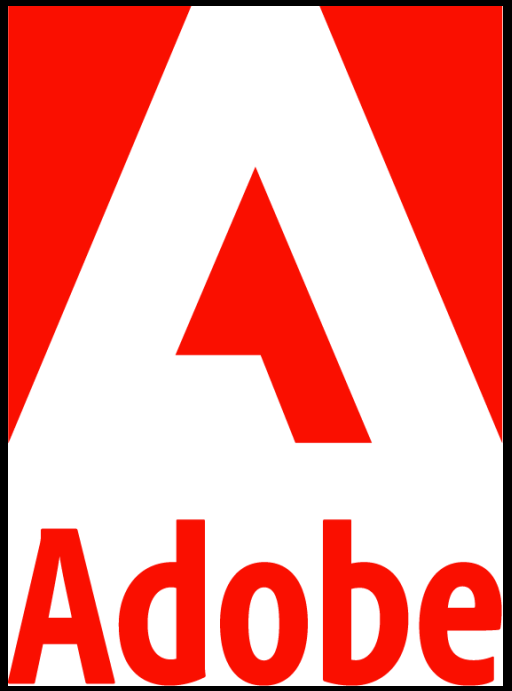
Alexander Stepanov

https://www.youtube.com/watch?v=1-CmNNp5eag

Connections between  

STL and Its Design Principles

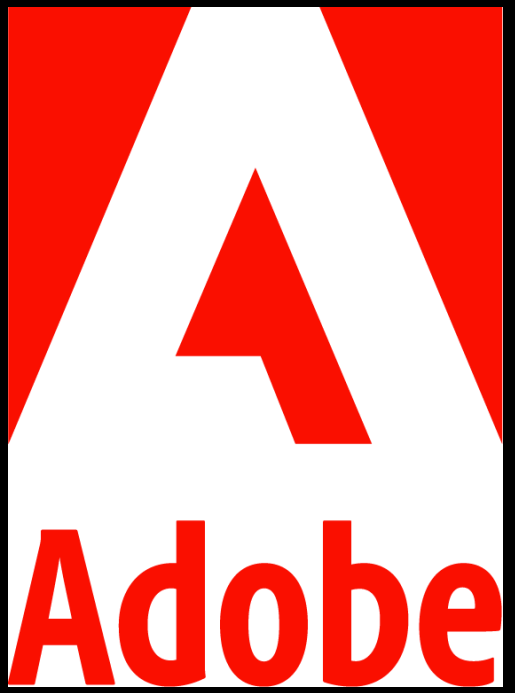
Alexander Stepanov



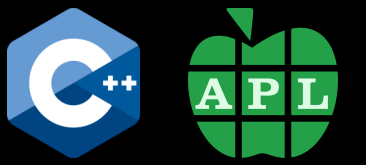
Connections between  

STL and Its Design Principles

Alexander Stepanov



2002

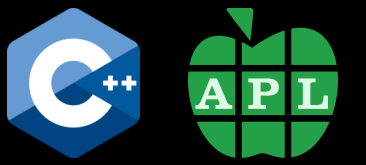
Connections between 

"Very clearly things like
`accumulate` (reduction)
comes from `APL`."

Alexander Stepanov

STL and Its Design Principles


FIFA TV

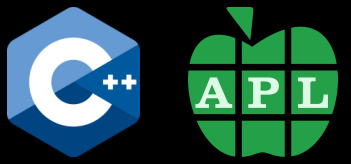
Connections between 

"Very clearly things like
`accumulate` (reduction)
comes from `APL`."

Alexander Stepanov

STL and Its Design Principles

Connections between



"There are a lot of things borrowed from APL."

Alexander Stepanov

STL and Its Design Principles

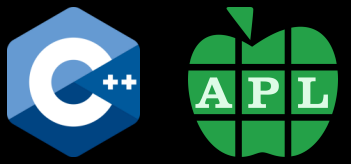
Connections between  

`accumulate`

Connections between


```
iota
accumulate
```

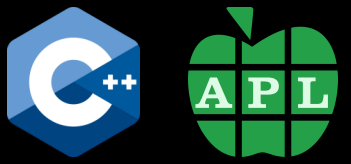
Connections between




```
#include <numeric>
```

iota
accumulate

Connections between




```
#include <numeric>
```

```
iota
accumulate
partial_sum?
```

1979 ACM Turing Award Lecture

Delivered at ACM '79, Detroit, Oct. 29, 1979

The 1979 ACM Turing Award was presented to Kenneth E. Iverson by Walter Carlson, Chairman of the Awards Committee, at the ACM Annual Conference in Detroit, Michigan, October 29, 1979.

In making its selection, the General Technical Achievement Award Committee cited Iverson for his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as APL. Iverson's contributions to the implementation of interactive systems, to the educational uses of APL, and to programming language theory and practice were also noted.

Born and raised in Canada, Iverson received his doctorate in 1954 from Harvard University. There he served as Assistant Professor of Applied Mathematics from 1955–1960. He then joined International Business Machines, Corp. and in 1970 was named an IBM Fellow in honor of his contribution to the development of APL.

Dr. Iverson is presently with I.P. Sharp Associates in Toronto. He has published numerous articles on programming languages and has written four books about programming and mathematics: *A Programming Language* (1962), *Elementary Functions* (1966), *Algebra: An Algorithmic Treatment* (1972), and *Elementary Analysis* (1976).

Notation as a Tool of Thought

Kenneth E. Iverson
IBM Thomas J. Watson Research Center

Key Words and Phrases: APL, mathematical notation
CR Category: 4.2

Author's present address: K.E. Iverson, I.P Sharp Associates, 145 King Street West, Toronto, Ontario, Canada M5H1J8.

The importance of nomenclature, notation, and language as tools of thought has long been recognized. In chemistry and in botany, for example, the establishment of systems of nomenclature by Lavoisier and Linnaeus did much to stimulate and to channel later investigation. Concerning language, George Boole in his *Laws of Thought* [1, p.24] asserted "That language is an instrument of human reason, and not merely a medium for the expression of thought, is a truth generally admitted."

Mathematical notation provides perhaps the best-known and best-developed example of language used consciously as a tool of thought. Recognition of the important role of notation in mathematics is clear from the quotations from mathematicians given in Cajori's *A History of Mathematical Notations* [2, pp.332,331]. They are well worth reading in full, but the following excerpts suggest the tone:

> By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.
>
> A.N. Whitehead

when applied to the argument ⍵, and the *sum reduction* denoted by +/ produces the sum of the elements of its vector argument, and will be shown as follows:

```
      ⍳5
1 2 3 4 5
      +/⍳5
15
```

We will use one non-executable bit of notation: the symbol ↔ appearing between two expressions asserts their equivalance.

1.1 Ease of Expressing Constructs Arising in Problems

If it is to be effective as a tool of thought, a notation must allow convenient expression not only of notions arising directly from a problem, but also of those arising in subsequent analysis, generalization, and specialization.

Consider, for example, the crystal structure illustrated by Figure 1, in which successive layers of atoms lie not directly on top of one another, but lie "close-packed" between those below them. The numbers of atoms in successive rows from the top in Figure 1 are therefore given by ⍳5, and the total number is given by +/⍳5.

The three-dimensional structure of such a crystal is also close-packed; the atoms in the plane lying above Figure 1 would lie between the atoms in the plane below it, and would have a base row of four atoms. The complete three-dimensional structure corresponding to Figure 1 is therefore a tetrahedron whose planes have bases of lengths 1, 2, 3, 4, and 5. The numbers in successive planes are therefore the *partial* sums of the vector ⍳5, that is, the sum of the first element, the sum of the first two elements, etc. Such partial sums of a vector ⍵ are denoted by +\⍵, the function +\ being called *sum scan*. Thus:

```
      +\⍳5
1 3 6 10 15
      +/+\⍳5
35
```

The final expression gives the total number of atoms in the tetrahedron.

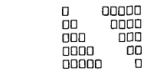
The sum +/⍳5 can be represented graphically in other ways, such as shown on the left of Figure 2. Combined with the inverted pattern on the right, this representation suggests that the sum may be simply related to the number of units in a rectangle, that is, to a product.

The lengths of the rows of the figure formed by pushing together the two parts of Figure 2 are given by adding the vector ⍳5 to the same vector reversed. Thus:

```
      ⍳5
1 2 3 4 5
      ⌽⍳5
5 4 3 2 1
      (⍳5)+(⌽⍳5)
6 6 6 6 6
```

Fig. 1.

Fig. 2.



This pattern of 5 repetitions of 6 may be expressed as 5⍴6, and we have:

```
      5⍴6
6 6 6 6 6
      +/5⍴6
30
      6×5
30
```

The fact that +/5⍴6 ↔ 6×5 follows from the definition of multiplication as repeated addition.

The foregoing suggests that +/⍳5 ↔ (6×5)÷2, and, more generally, that:

```
      +/⍳N ↔ ((N+1)×N)÷2                                    A.1
```

1.2 Suggestivity

A notation will be said to be *suggestive* if the forms of the expressions arising in one set of problems suggest related expressions which find application in other problems. We will now consider related uses of the functions introduced thus far, namely:

```
      ⍳   ⌽   ⍴   +/   +\
```

The example:

```
      5⍴2
2 2 2 2 2
      ×/5⍴2
32
```

suggests that ×/M⍴N ↔ N*M, where * represents the power function. The similiarity between the definitions of power in terms of times, and of times in terms of plus may therefore be exhibited as follows:

```
      ×/M⍴N ↔ N*M
      +/M⍴N ↔ N×M
```

Similar expressions for partial sums and partial products may be developed as follows:

```
      ×\5⍴2
2 4 8 16 32
      2*⍳5
2 4 8 16 32

      ×\M⍴N ↔ N*⍳M
      +\M⍴N ↔ N×⍳M
```

Because they can be represented by a triangle as in Figure 1, the sums +\⍳5 are called *triangular* numbers. They are a special case of the *figurate* numbers obtained by repeated applications of sum scan, beginning either with +\⍳N, or with +\N⍴1. Thus:

```
      5⍴1
1 1 1 1 1

      +\5⍴1
1 2 3 4 5

      +\+\5⍴1
1 3 6 10 15

      +\+\+\5⍴1
1 4 10 20 35
```

lie "close-packed" between those below them. The numbers of atoms in successive rows from the top in Figure 1 are therefore given by ⍳5, and the total number is given by +/⍳5.

The three-dimensional structure of such a crystal is also close-packed; the atoms in the plane lying above Figure 1 would lie between the atoms in the plane below it, and would have a base row of four atoms. The complete three-dimensional structure corresponding to Figure 1 is therefore a tetrahedron whose planes have bases of lengths 1, 2, 3, 4, and 5. The numbers in successive planes are therefore the *partial* sums of the vector ⍳5, that is, the sum of the first element, the sum of the first two elements, etc. Such partial sums of a vector *V* are denoted by +\V, the function +\ being called *sum scan.* Thus:

```
      +\⍳5
1 3 6 10 15
      +/+\⍳5
35
```

forms of the expre
lems suggest relat
cation in other p
related uses of th
namely:

```
      ⍳   ⌽   ⍴   +
```

The example:

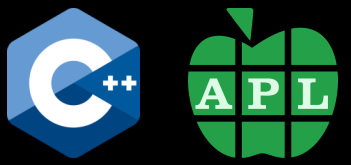
```
        5⍴2
2 2 2 2 2
        ×/5⍴2
32
```

suggests that ×/M⍴
power function. T
nitions of power i
terms of plus ma
lows:

×/M⍴N ↔ N*M
+/M⍴N ↔ N×M

Similar expressio

Connections between




```
#include <numeric>
```

iota
accumulate
partial_sum?

tries in a truth table for three boolean arguments. The general expression for N elements is easily seen to be `(Nρ2)⊤(⍳2*N)-1`, and we may wish to assign an ad hoc name to this function. Using the direct definition form (Appendix B), the name T is assigned to this function as follows:

```
T:(ωρ2)⊤(⍳2*ω)-1                                    A.2
```

The symbol ω represents the argument of the function; in the case of two arguments the left is represented by α. Following such a definition of the function T, the expression `T 3` yields the boolean matrix BN shown above.

Three expressions, separated by colons, are also used to define a function as follows: the middle expression is executed first; if its value is zero the first expression is executed, if not, the last expression is executed. This form is convenient for recursive definitions, in which the function is used in its own definition. For example, a function which produces binomial coefficients of an order specified by its argument may be defined recursively as follows:

```
BC:(X,0)+(0,X←BC ω-1):ω=0:1                         A.3
```

Thus `BC 0` ↔ `1` and `BC 1` ↔ `1 1` and `BC 4` ↔ `1 4 6 4 1`.

The term *operator*, used in the strict sense defined in mathematics rather than loosely as a synonym for *function*, refers to an entity which applies to functions to produce functions; an example is the derivative operator.

We have already met two operators, *reduction*, and *scan*, denoted by `/` and `\`, and seen how they contribute to brevity by applying to different functions to produce families of related functions such as `+/` and `×/` and `∧/`. We will now illustrate the notion further by introducing the *inner product* operator denoted by a period. A function (such as `+/`) produced by an operator will be called a *derived* function.

If P and Q are two vectors, then the inner product `+.×` is defined by:

```
P+.×Q ↔ +/P×Q
```

and analogous definitions hold for function pairs other than `+` and `×`. For example:

```
      P←2 3 5
      Q←2 1 2
      P+.×Q
17
      P×.*Q
300
      Pl.+Q
4
```

Each of the foregoing expressions has at least one useful interpretation: `P+.×Q` is the total cost of order quantities Q for items whose prices are given by P; because P is a vector of primes, `P×.*Q` is the number whose prime decomposition is given by the exponents Q; and if P gives distances from a source to transhipment points and Q gives distances from the transhipment points to the destination, then `Pl.+Q` gives the minimum distance possible.

The function `+.×` is equivalent to the inner product or dot product of mathematics, and is extended to matrices as in mathematics. Other cases such as `×.*` are extended analogously. For example, if T is the function defined by A.2, then:

```
      T 3
0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1
      P+.×T 3
0 5 3 8 2 7 5 10
      P×.*T 3
1 5 3 15 2 10 6 30
```

These examples bring out an important point: if B is boolean, then `P+.×B` produces sums over subsets of P specified by `1`'s in B, and `P×.*B` produces products over subsets.

The phrase `∘.×` is a special use of the inner product operator to produce a derived function which yields products of each element of its left argument with each element of its right. For example:

```
      2 3 5∘.×⍳5
 2  4  6  8 10
 3  6  9 12 15
 5 10 15 20 25
```

The function `∘.×` is called *outer product*, as it is in tensor analysis, and functions such as `∘.+` and `∘.*` and `∘.<` are defined analogously, producing "function tables" for the particular functions. For example:

```
      D←0 1 2 3
      D∘.⌈D
0 1 2 3
1 1 2 3
2 2 2 3
3 3 3 3
      D∘.≥D
1 0 0 0
1 1 0 0
1 1 1 0
1 1 1 1
      D∘.!D
1 1 1 1
0 1 2 3
0 0 1 3
0 0 0 1
```

The symbol `!` denotes the binomial coefficient function, and the table `D∘.!D` is seen to contain Pascal's triangle with its apex at the left; if extended to negative arguments (as with `D←¯3 ¯2 ¯1 0 1 2 3`) it will be seen to contain the triangular and higher-order figurate numbers as well. This extension to negative arguments is interesting for other functions as well. For example, the table `D∘.×D` consists of four quadrants separated by a row and a column of zeros, the quadrants showing clearly the rule of signs for multiplication.

Patterns in these function tables exhibit other properties of the functions, allowing brief statements of proofs by exhaustion. For example, commutativity appears as a symmetry about the diagonal. More precisely, if the result of the transpose function `⍉` (which reverses the order of the axes of its argument) applied to a table `T←D∘.⌈D` agrees with T, then the function `⌈` is commutative on the domain. For example, `T=⍉T←D∘.⌈D` produces a table of `1`'s because `⌈` is commutative.

defined in mathematics rather than loosely as a synonym for *function*, refers to an entity which applies to functions to produce functions; an example is the derivative operator.

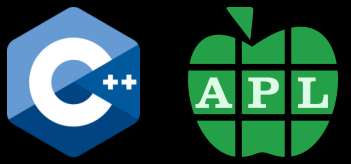
We have already met two operators, *reduction*, and *scan*, denoted by / and \, and seen how they contribute to brevity by applying to different functions to produce families of related functions such as +/ and ×/ and ∧/. We will now illustrate the notion further by introducing the *inner product* operator denoted by a period. A function (such as +/) produced by an operator will be called a *derived* function.

If P and Q are two vectors, then the inner product +.× is defined by:

```
P+.×Q ↔ +/P×Q
```

and analogous definitions hold for function pairs

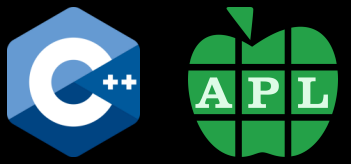
Connections between




```
#include <numeric>
```

iota
accumulate
partial_sum

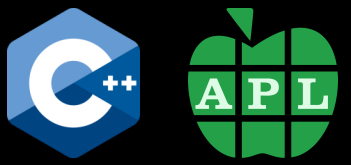
Connections between




```
#include <numeric>
```

```
iota
accumulate
partial_sum
inner_product
```

Connections between




```
#include <numeric>
```

```
iota
accumulate
partial_sum
inner_product
adjacent_difference
```

Connections between

```
iota           ← ⍳
accumulate     ← /
partial_sum    ← \
inner_product  ← .
adjacent_difference ← 2-/
```

Connections between

```
iota           ← ⍳
accumulate     ← /
partial_sum    ← \
inner_product  ← .
adjacent_difference ← 2-⍨/
```

Connections between


```
iota                ← ⍳
accumulate          ← +⌿
partial_sum         ← +\
inner_product       ← +.×
adjacent_difference ← 2-⍨⌿
```

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
accumulate reduce[17]	1	Yes	Reduce	plus{}
	count, count_if, min_element, max_element, minmax_element			
adjacent_reduce	2	Yes	Reduce	-
partial_sum inclusive_scan[17]	1	Yes	Map	plus{}
adjacent_inclusive_scan	2	Yes	Map	-
find_if	1/2	No	Reduce	-
	find, all_of, any_of, none_of			
adjacent_find_if	2	No	Reduce	equal{}
transform	1/2	No	Map	-
	replace[17], replace_if[17]			
adjacent_transform	2	No	Map	-
inner_product transform_reduce[17]	1/2	Yes	Reduce	plus{} multiplies{}
transform_inclusive_scan[17]	1/2	Yes	Map	-

Note: non-accumulator reductions all short-circuit

THE ALGORITHM INTUITION TABLE

Algorithm	Indexes Viewed	Accumulator	Reduce / Map	Default Op
reduce[17]	1	Yes	Reduce	-
	count, count_if, min_element, max_element, minmax_element			
adjacent_reduce	2	Yes	Reduce	-
scan[17]	1	Yes	Map	-
adjacent_scan	2	Yes	Map	-
find_if	1/2	No	Reduce	-
	find, all_of, any_of, none_of			
adjacent_find_if	2	No	Reduce	-
transform	1/2	No	Map	-
	replace[17], replace_if[17]			
adjacent_transform	2	No	Map	-
transform_reduce[17]	1/2	Yes	Reduce	-
transform_scan[17]	1/2	Yes	Map	-

Note: non-accumulator reductions all short-circuit

Genesis of STL

- Original intuition: associating algorithms with mathematical theories – 1976
- Specification language *Tecton* (with Dave Musser and Deepak Kapur) – 1979 to 1983
- Higher-order programming in Scheme (with Aaron Kershenbaum and Dave Musser) – 1984 to 1986
- Ada Generic Library (with Dave Musser) – 1986
- UKL Standard Components – 1987 to 1988
- STL (with Meng Lee and Dave Musser) – 1993 to 1994
- SGI STL (with Matt Austern and Hans Boehm) – 1995 to 1998

http://stepanovpapers.com/
http://stepanovpapers.com/schemenotes/notes.pdf

```
(define (for-each-in-interval first last)
  (iterator-until
    (bind-1-of-2 < last)
    (primitive-iterator first 1+)
    'will-never-use-this-marker))
```

it would also be nice to implement reduction (reduction operator was introduced by Kenneth Iverson in APL)

```
(define (reduce iterator)
  (lambda (function . initial-value)
    (define (add-to x)
      (set! initial-value (function initial-value x)))
    (cond (initial-value
           (set! initial-value (car initial-value))
           (iterator add-to)
           initial-value)
          (else
           (let ((marker #!false))
             (define (first-time x)
               (set! initial-value x)
               (set! marker #!true)
               (set! first-time add-to))
             (iterator (lambda (x) (first-time x)))
             (if marker initial-value (function)))))))
```

where set! is a special form that changes a value of a binding

Connections between


```
iota
reduce
partial_sum
inner_product
*_scan
ceil
floor
```

Connections between

Barry Revzin @BarryRevzin · Dec 28, 2018

People really hate the name iota(). Lots of people calling it the worst name

Barry Revzin
@BarryRevzin

Replying to @ben_deane

I'm guessing it's 0, to a first approximation. But the nice thing about iota (which is also true for like... car and cdr) is that while it conveys ~no info about what it does, once you learn it... you know it cold forever?

9:12 PM · Dec 28, 2018 · Twitter for Android

---

1 Like

8:04 PM · Dec 28, 2018 · Twitter for Android

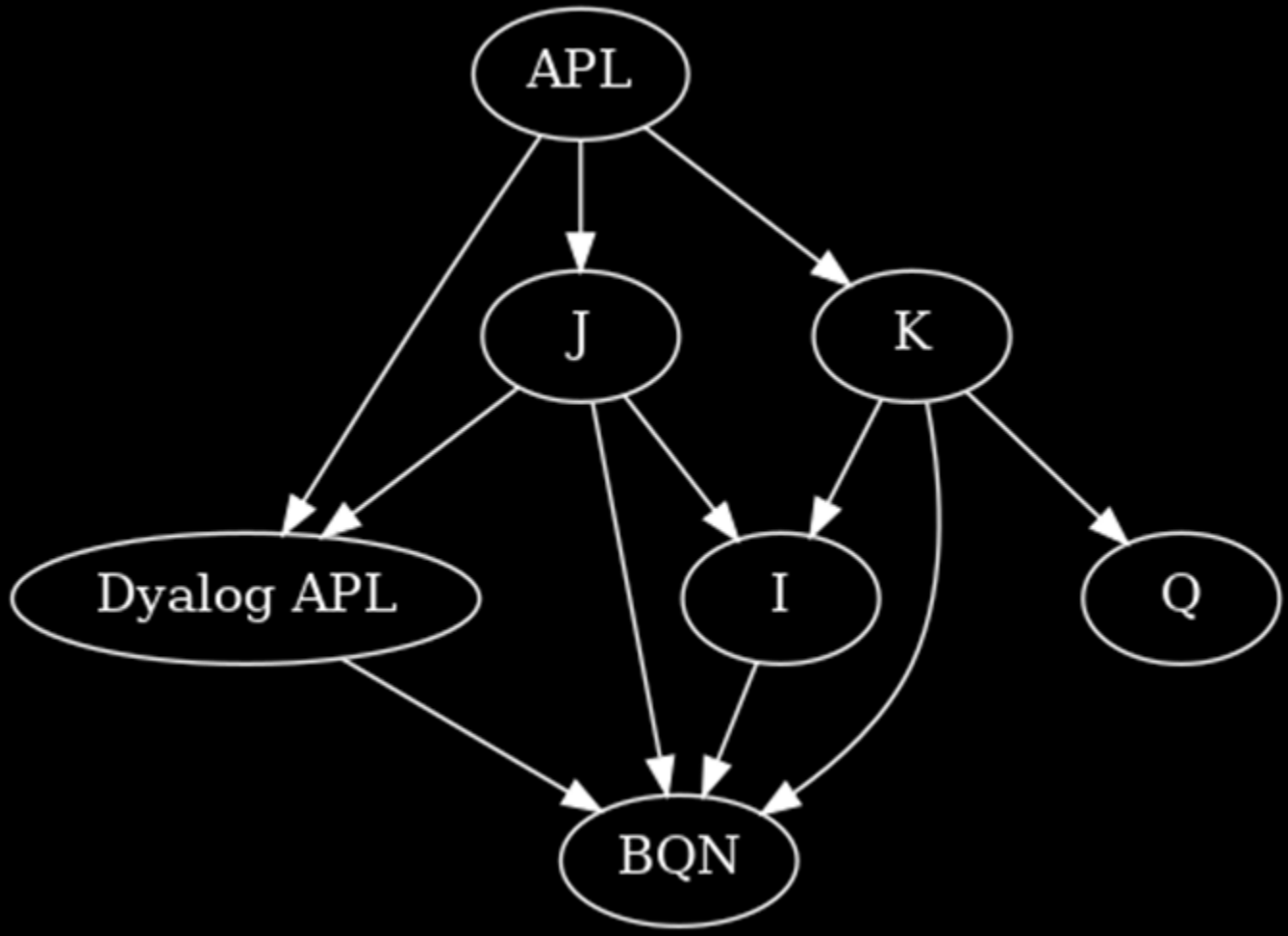
---

1 Retweet **3** Likes

History

History

(& mini-histories / random facts)


APL
J
K
Dyalog APL
I
Q
BQN

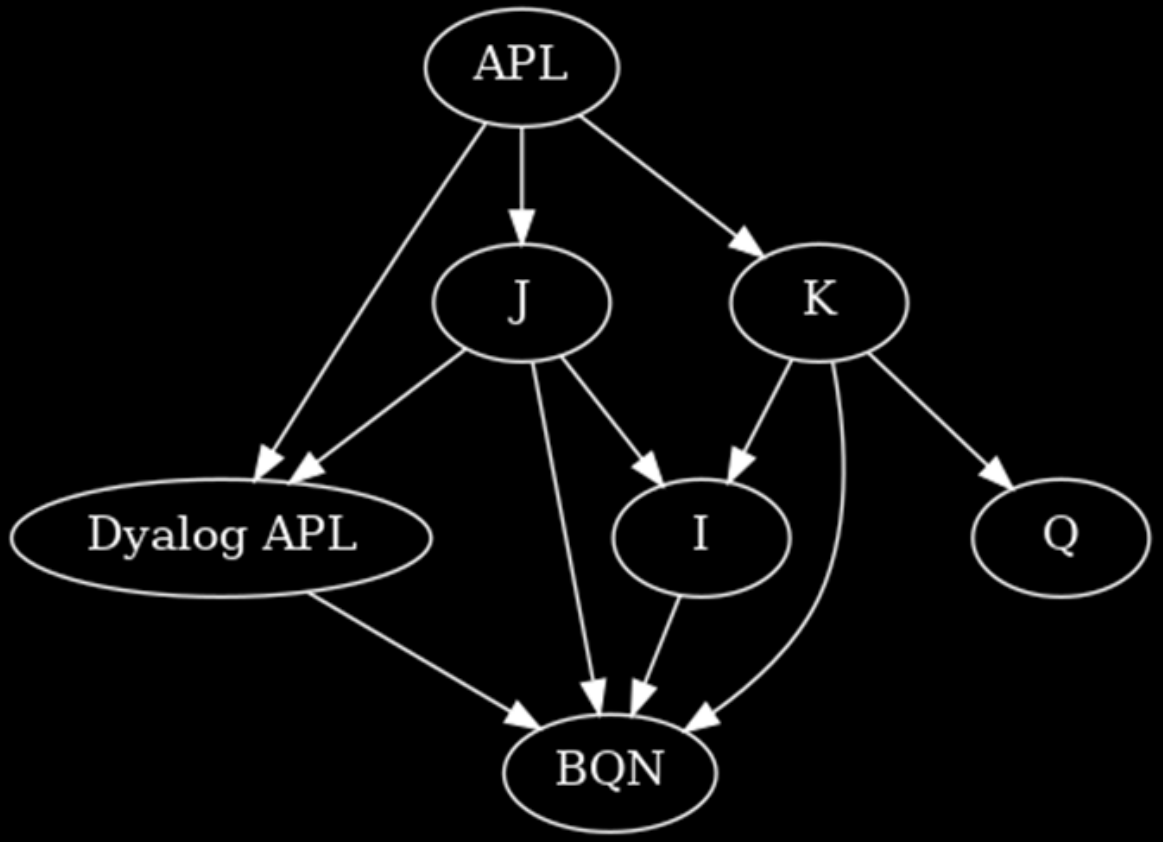


Table 1. Array languages timeline.

Language	Year
APL	1966
Dyalog APL 1.0	1983
J	1990
K	1994
Q	2003
I	2012
BQN	2020
Dyalog APL 18.0	2020

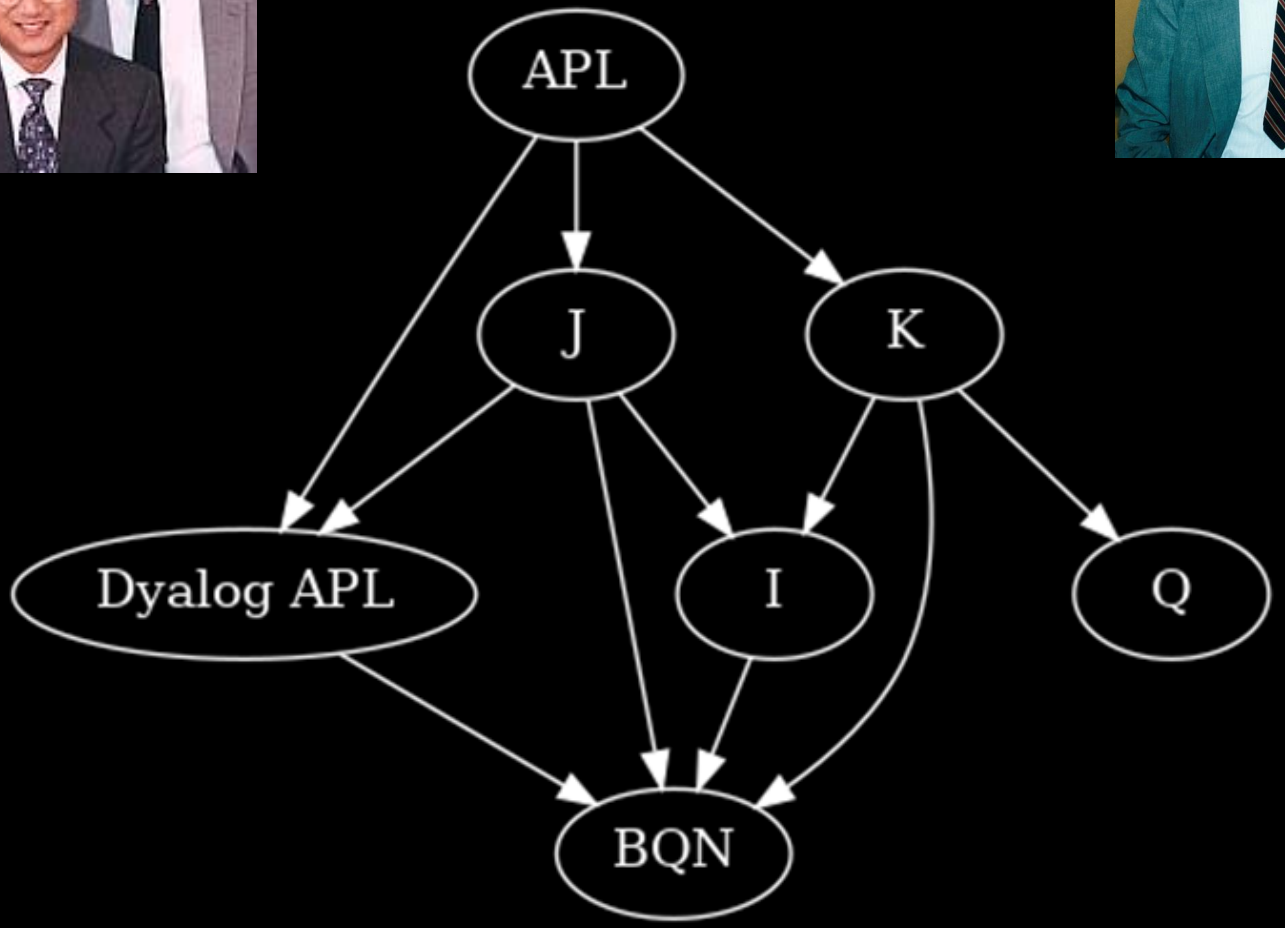


Table 1. Array languages timeline.

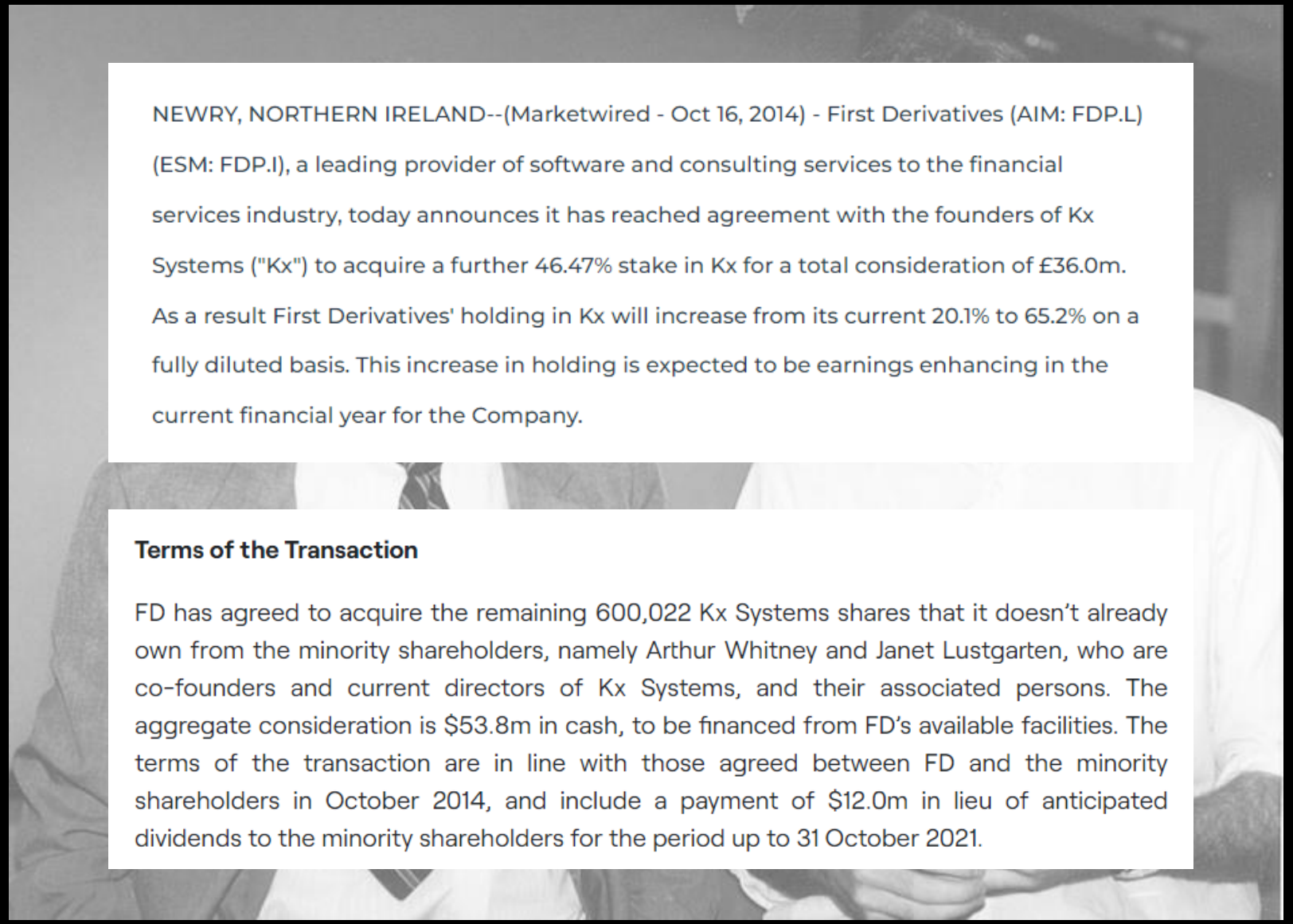
Language	Year
APL	1966
Dyalog APL 1.0	1983
J	1990
K	1994
Q	2003
I	2012
BQN	2020
Dyalog APL 18.0	2020

Arthur Whitney & kx

Mini History 1

A
A+
K
q
Shakti

A
A+
K0
K1
K2
K3
K4
q
K5
K6
K7
K9
Shakti

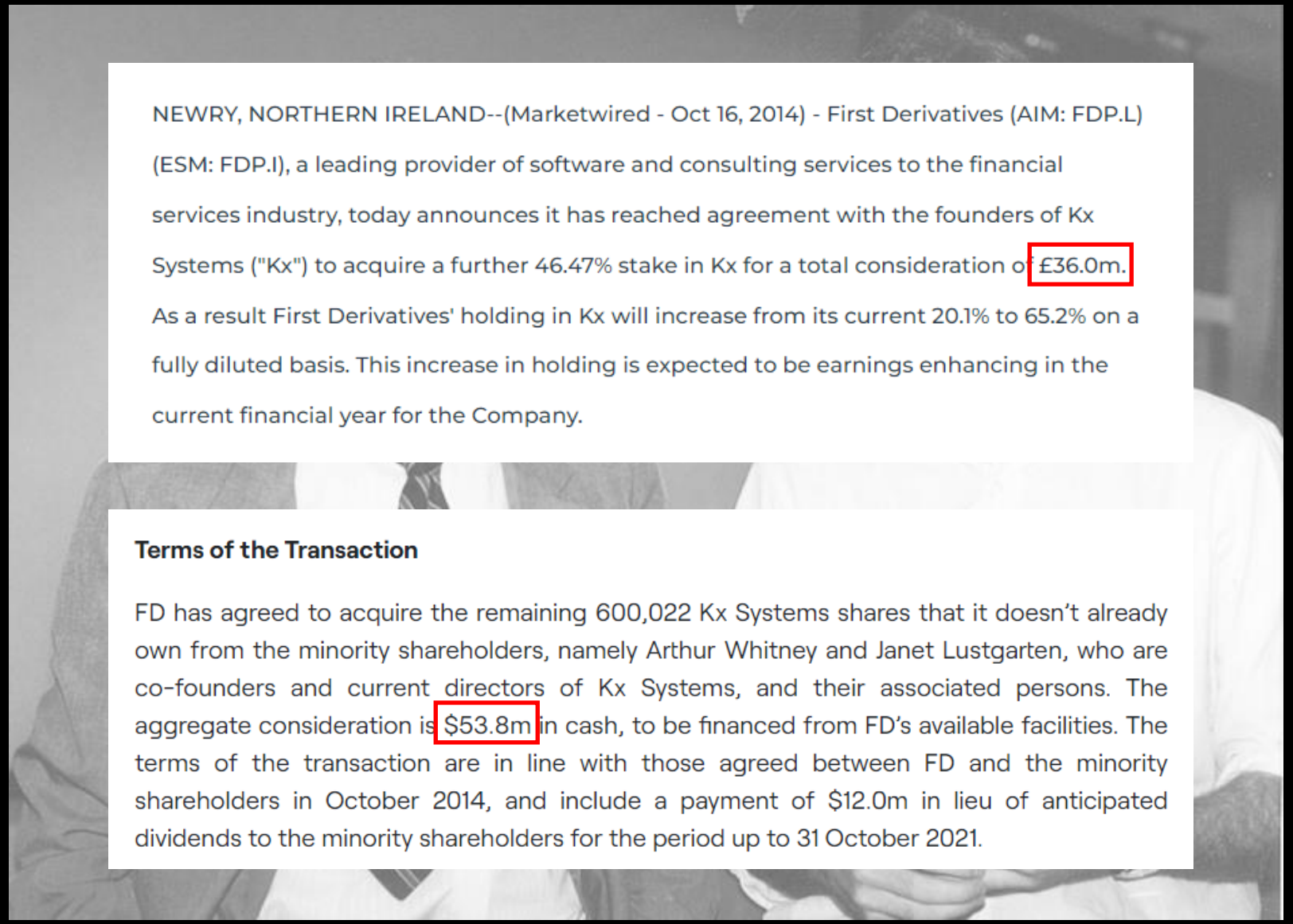


NEWRY, NORTHERN IRELAND--(Marketwired - Oct 16, 2014) - First Derivatives (AIM: FDP.L) (ESM: FDP.I), a leading provider of software and consulting services to the financial services industry, today announces it has reached agreement with the founders of Kx Systems ("Kx") to acquire a further 46.47% stake in Kx for a total consideration of £36.0m. As a result First Derivatives' holding in Kx will increase from its current 20.1% to 65.2% on a fully diluted basis. This increase in holding is expected to be earnings enhancing in the current financial year for the Company.

Terms of the Transaction

FD has agreed to acquire the remaining 600,022 Kx Systems shares that it doesn't already own from the minority shareholders, namely Arthur Whitney and Janet Lustgarten, who are co-founders and current directors of Kx Systems, and their associated persons. The aggregate consideration is $53.8m in cash, to be financed from FD's available facilities. The terms of the transaction are in line with those agreed between FD and the minority shareholders in October 2014, and include a payment of $12.0m in lieu of anticipated dividends to the minority shareholders for the period up to 31 October 2021.

A
A+
K0
K1
K2
K3
K4
q
K5
K6
K7
K9
Shakti



NEWRY, NORTHERN IRELAND--(Marketwired - Oct 16, 2014) - First Derivatives (AIM: FDP.L) (ESM: FDP.I), a leading provider of software and consulting services to the financial services industry, today announces it has reached agreement with the founders of Kx Systems ("Kx") to acquire a further 46.47% stake in Kx for a total consideration of £36.0m. As a result First Derivatives' holding in Kx will increase from its current 20.1% to 65.2% on a fully diluted basis. This increase in holding is expected to be earnings enhancing in the current financial year for the Company.

Terms of the Transaction

FD has agreed to acquire the remaining 600,022 Kx Systems shares that it doesn't already own from the minority shareholders, namely Arthur Whitney and Janet Lustgarten, who are co-founders and current directors of Kx Systems, and their associated persons. The aggregate consideration is $53.8m in cash, to be financed from FD's available facilities. The terms of the transaction are in line with those agreed between FD and the minority shareholders in October 2014, and include a payment of $12.0m in lieu of anticipated dividends to the minority shareholders for the period up to 31 October 2021.

A
A+
K0
K1
K2
K3
K4
q
K5
K6
K7
K9
Shakti

5100 & Bill Gates

Mini History 2


BRIGHTNESS
L32 64 R32
IN PROCESS
POWER ON
PROCESS CHECK
POWER OFF
IBM 5100
Portable Computer
REVERSE DISPLAY
BASIC
APL
RESTART
DISPLAY REGISTERS
NORMAL
CMD
EXECUTE
ATTN
HOLD


POWER ON
PROCESS CHECK
POWER OFF
IBM 5100
Portable Computer
APL
BASIC
LOAD
SAVE
RUN
GO
LIST
UTIL
AUTO
RENUM
RD=
REWIND
Calc Result
Copy Display
Delete
Insert
EXECUTE
CMD
ATTN
HOLD
CHAIN
CLOSE
DATA
DEF
DIM
END
FNEND
FOR
GET
GOSUB
GOTO
IF
INPUT
LET
MAT
NEXT
OPEN
PAUSE
PRINT
READ
REM
RESET
RESTR
RETRN
STOP
USE
USING


BRIGHTNESS
L32 64 R32
IN PROCESS
PROCESS CHECK
POWER ON
POWER OFF
IBM 5100
Portable Computer
REVERSE DISPLAY
BASIC
APL
RESTART
DISPLAY REGISTERS
NORMAL
LIST
UTIL
AUTO
RENUM
RD=
REWIND
Calc Result
Copy Display
Delete
Insert
EXECUTE
ATTN
HOLD


REVERSE DISPLAY
BASIC
APL
RESTART

Bill Gates

https://www.jsoftware.com/papers/APLQA.htm

"**Bill Gates** had recently dropped out of Harvard and was keen on the possibilities of the coming wave of personal computers. This was before the IBM PC, but not much before. He had some knowledge of **APL** and thought of it as a possible language for the new machines. He was traveling across North America talking to anyone who would talk to him. Ian was on his list because of **APL**. Ian talked with lots of people who had crazy ideas. Gates happened to be the one who turned out to be right.

He cold-called Ian and asked for a meeting. Shortly after that meeting started Ian called me and asked if I would join them. The three of us chatted pleasantly in Ian's office for some time. Then I gave Gates a tour of the data center and the development offices. I introduced him to whomever happened to be around and that included Bob. As far as I remember that was the end of it."

Eric Iverson

APL Quotations and Anecdotes

"**Gates** came into my office, and we talked a bit about development work. I asked him where they were going with **APL**. He said that they had tried building an **APL** interpreter, but gave up on the idea, because it spent all its time looking up names in the symbol table. I told him that was precisely why nobody did it that way, instead tokenizing **APL** code so that such time-consuming actions were performed only once. He shrugged that off, and switched the topic to his new project, a graphical front end to DOS, based on the Xerox work. That front end was Windows, of course."

Bob Bernecky

APL Quotations and Anecdotes

"APL could have become a dominant language for the microcomputer if only some software company had developed an APL interpreter for early microcomputers based on popular microprocessors such as the Intel 8080 or Zilog Z80. In the 1970s, those who identified microcomputers with BASIC and viewed that language as an insignificant programming toy, could have entered the microcomputer market earlier by choosing an APL-based machine. In view of the popularity of APL, such a company could have derived substantial profits from selling its APL software — the same way Microsoft was paying its bills with the sales of BASIC. In fact it was Microsoft's co-founder and APL's vocal supporter Bill Gates who wanted to do just that. Microsoft had been developing its own APL interpreter since 1976, perhaps under the influence of IBM's introduction of its APL-based IBM 5100 computer. "Equivalence with the 5100 was my goal," explained Gates in his March 1979 interview for ETI Canada ["APL: Good for the Brain"]. After the success of Microsoft BASIC, first offered to the owners of the Altair 8800 microcomputers in mid-1975, "APL seemed like a great follow-on product," continued Gates. However, in 1976, Microsoft was still a small software company carefully addressing market needs. FORTRAN got higher priority and by the end of 1976, only one person was left at Microsoft to continue the APL work. Microsoft FORTRAN was introduced in June 1977 and COBOL the following year. It was not until 1979 that Microsoft announced its APL-80 interpreter for the Intel 8080 and Zilog Z80 platforms. It was to be out in April 1979 and compatible with IBM's APL.SV software. But in the end the Microsoft APL-80 proved to be vaporware and by the early 1980s, it was Microsoft's BASIC and not APL that was installed on the majority of personal computers."

Zbigniew Stachniak

Inventing the PC, 2011, p. 184.

APL: Good For The Brain

A Programming Language has been around for many years, yet it is fast becoming THE new language. This exciting development in small computing systems is something that we think will be good for all of us.

IT HAS BEEN observed that not only does a language (English, French etc!) provide a means of expressing thoughts, it also tends to limit thoughts, to those expressible in the language whether to others, or to one's self. Just as a road provides a path for transportation, if you get used to travelling by car you kind of have to stick to the road, and the surrounding terrain remains undiscovered.

So it is with computer languages. Iverson Notation was developed for humans to use, to advance their ability to think and solve problems. APL grew from this, and is probably the only language to pick up an already developed notation, as opposed to acting as a compromise between English and machine code.

Thus while other languages tend frequently to constrict one in trying to solve a problem, learning and using APL tends to help one to handle the problem. In fact with experience one seems to be able to write an APL program while thinking up the method of solution. Other languages usually require an intermediate block diagram or flow chart stage for a similar problem. The proof of this lies in the absolutely fanatical following for APL, and the well attended and highly regarded APL conferences which take place to discuss applications and possible new features for the language.

We think APL is great for the mind and body, and that it's about to arrive in a big way. This article deals with what it is, where it's been, and a look at when it should be used, and where it's going.

WHAT IT'S LIKE

In as much as APL is quite different from other computer languages, has vastly different qualities, and has fanatical supporters, it is very useful to know something about the language itself.

APL having originally evolved from Iverson Notation, it is most helpful to start with those features which are actually part of the notation.

IVERSON NOTATION

The first step to getting a grasp on Iverson Notation is to be familiar with a few 'buzz-concepts' essential to the topic.

Essentially Iverson Notation is a mathematical notation which permits expressions to be written in very compact and condensed forms. The idea is that by giving a lot of the most used functions single symbol names (instead of just + − × and ÷) the user can not only more easily write his ideas down, but even learn to think on a higher level. A good example of this is a sorting operation, for example putting a list of names in alphabetical order. Iverson Notation gives you the concepts of how to think of the list of names, and then allows you to write down, using just a couple of symbols, the method used to do it. Example later.

DATA TYPE

In a mathematical expression, one may have a variety of different types

APL Applied

APL's big strength stems from the ability of a person familiar with the language to write a program as fast as thinking of the method. In fact one even tends to think of the solution in APL form. The other influential feature is that APL is an interactive language.

These two combine to produce an ideal way to use a computer for applications where the programs need to be ready fast, and where the applications area may be new (and hence the program is experimental and may require much revision). This is not to say that in other applications APL is not suitable, but until recently APL has been thought of as expensive in terms of execution time and memory use, or even simply to obtain access. In addition compared to other languages there are relatively few experienced APL programmers, the number familiar with the language will of course grow.

One estimate holds that an APL programmer can produce a program on the order of 25 times the speed of a COBOL programmer, (including debugging) and that the result is of course a far more compact listing.

One point to watch, however, is that while almost anyone can plough though a BASIC, FORTRAN or COBOL program (with enough patience), it is possible to program in APL in such a dense and tricky manner as to make the program virtually impossible to decipher by other persons. Programmers who do this also enjoy reducing 100 line FORTRAN programs to an APL one-liner. (There seems to be some special attraction to the 'one-line' challenge!)

However, if a program is written with reading also in mind, with a bit of descriptive documentation, there is no problem.

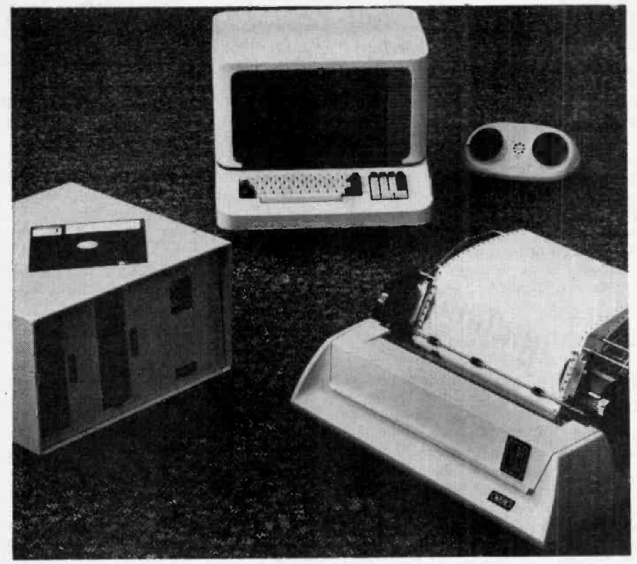


Fig. 5. Here's what the MCM 900 system looks like. This collection costs around $20,000. The keyboard-display unit contains the guts. The display itself is a 12 inch screen with 21 lines by 96 characters.

can tear himself away from the console!

VIDEOBRAIN

First out with APL appears to be Umtech with their F8 based Video Brain. Fitting into 13k - worth of ROM-in-a-cartridge, APL/S is a subset of 'full APL', although there is no standard APL. While a standard for APL is being worked on, big machine versions provide a reference which differs only slightly from one to another, and then only in housekeeping facilities rather than in the basic notation.

Z80 AND VANGUARD

Vanguard Systems Corp. have announced an APL interpreter on floppy disk for Z80, and of course all companies with Z80 based machines are eagerly waiting for it. It is reported to be 27K.

MICROSOFT

Meanwhile, over at Microsoft, much brain work is going into the development of APL interpreters for all kinds of processors, past present and future, ie 8080, Z80 and the 16 bit micros 8086, Z8000, and 68000. Bill Gates, president and APL Product Manager at Microsoft figures fall '79 will be the time his 8080 and Z80 interpreters will be ready with the 16 bit versions in early 1980. Vanguard are hoping to manage the same feat.

IT'S COMING!

Well, it's almost here. It appears that the development of APL on a 16 bit microprocessor will make the first really comfortable implementation of APL for personal, home, business or educational use. Because of this general purpose nature most new APL machines will be easy to use and hence either contain APL on ROM (fool proof) or on disk (most machines will have a disk or two anyhow).

Where Is APL Going?

Microsoft's Bill Gates has a bit to say.

Microsoft is a name in the microcomputer software field regarded with a great deal of awe. The US trade magazine 'Electronics' describes Microsoft as a 'small...software house', but with customers like Radio Shack and Commodore (TRS-80 Level II and PET BASICs) who are the leaders in consumerized home computing, and a huge list of other microcomputer companies, Microsoft has to be *the* major 'force' in micro software.

Bill Gates is the president of the company, and the APL job is largely his baby. So we asked him for his views on the micro APL scene.

NEAR FUTURE

What is about to happen in the way of micro APL?

'APL will see an incredible increase in popularity because it will be exposed to so many new people. To date, it has been an expensive language to use and is almost never introduced to first time computer users. APL's strengths assure that a significant percentage of personal computer users will adopt it as "the language". However it is not the ideal first time language and has some limitations of its own. BASIC will continue to dominate in this role, although specialized languages will be supported by personal computer manufacturers. Microsoft will introduce APL on the TRS-80, Exidy Sorcerer, Interact One, Nascom and NEC TK-80 in 1979.

Also the old time APL users will enjoy the low cost and ease of access to the low priced machines. These old

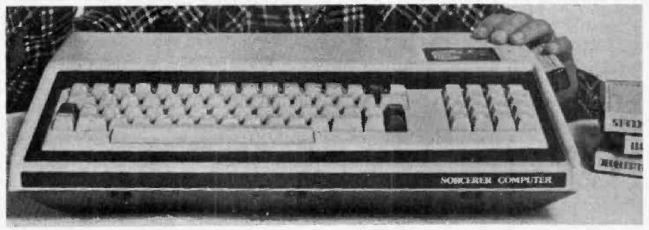
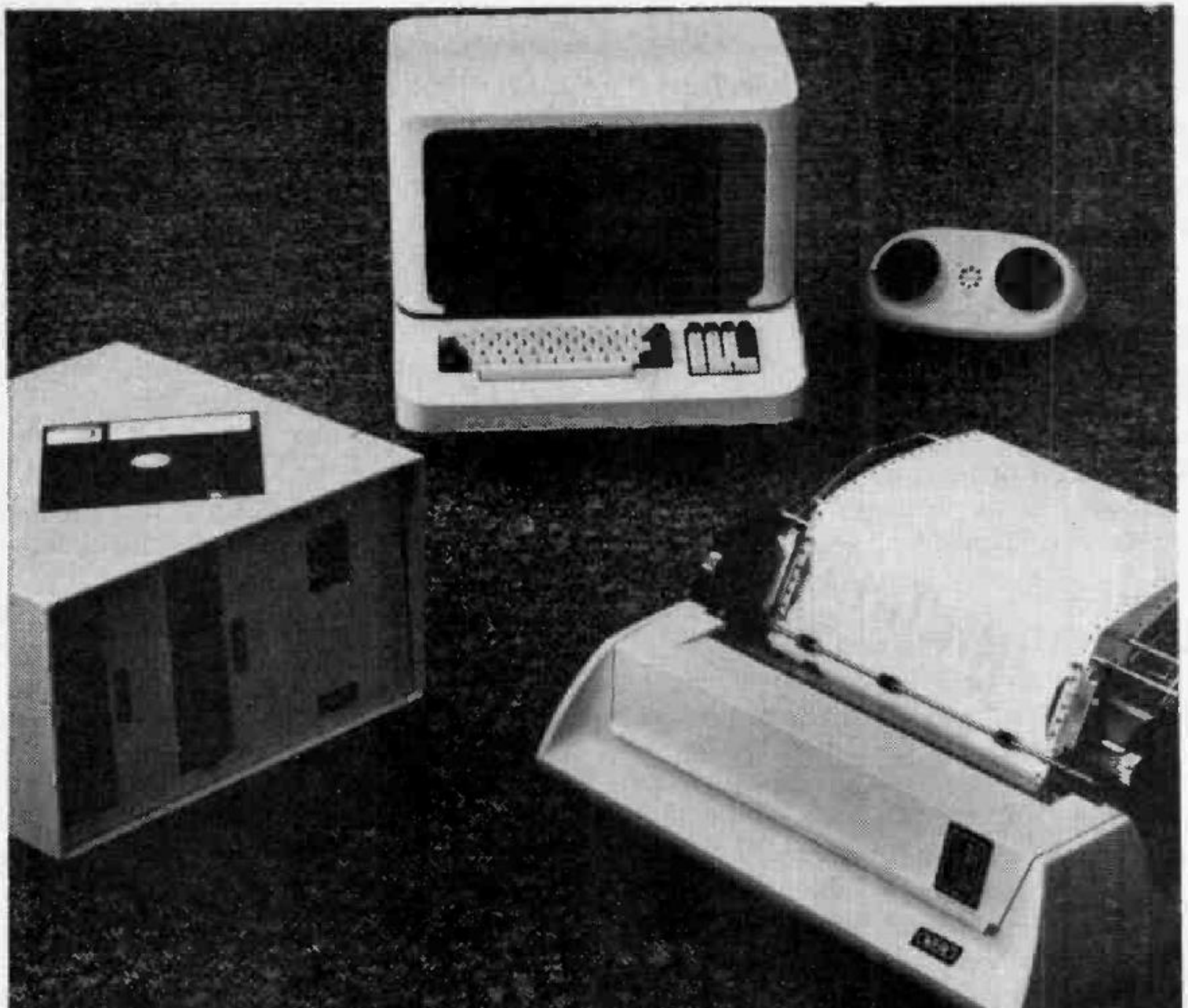


Fig. 6. This is the Exidy Sorcerer, which currently runs BASIC language. Plug in ROM-PACs, just visible to the right, will allow the use of other languages, with APL to be ready in the fall.



Where Is APL Going?

Microsoft's Bill Gates has a bit to say.

Microsoft is a name in the microcomputer software field regarded with a great deal of awe. The US trade magazine 'Electronics' describes Microsoft as a 'small...software house', but with customers like Radio Shack and Commodore (TRS-80 Level II and PET BASICs) who are the leaders in consumerized home computing, and a huge list of other microcomputer companies, Microsoft has to be *the* major 'force' in micro software.

Bill Gates is the president of the company, and the APL job is largely his baby. So we asked him for his views on the micro APL scene.

NEAR FUTURE

What is about to happen in the way of

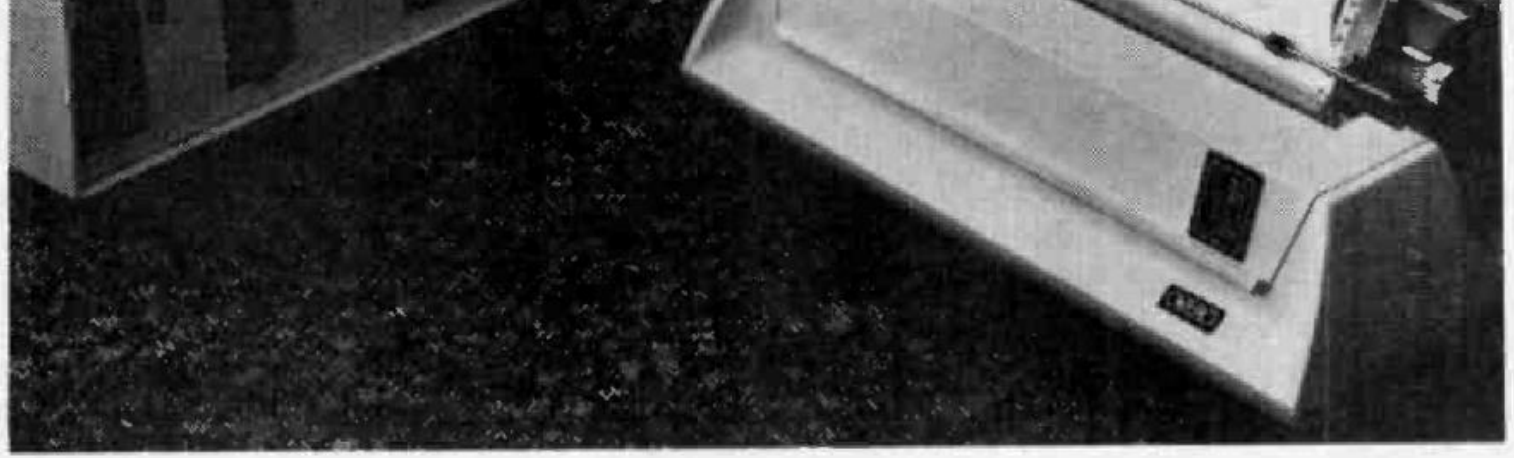


Fig. 5. Here's what the MCM 900 system looks like. This collection costs around $20,000. The keyboard-display unit contains the guts. The display itself is a 12 inch screen with 21 lines by 96 characters.

can tear himself away from the console!

VIDEOBRAIN

First out with APL appears to be Umtech with their F8 based Video Brain. Fitting into 13k - worth of ROM-in-a-cartridge, APL/S is a subset of 'full APL', although there is no standard APL. While a standard for APL is being worked on, big machine versions provide a reference which differs only slightly from one to another, and then only in housekeeping facilities rather than in the basic notation.

Z80 AND VANGUARD

Vanguard Systems Corp. have an-

will be ready with the 16 bit versions in early 1980. Vanguard are hoping to manage the same feat.

IT'S COMING!

Well, it's almost here. It appears that the development of APL on a 16 bit microprocessor will make the first really comfortable implementation of APL for personal, home, business or educational use. Because of this general purpose nature most new APL machines will be easy to use and hence either contain APL on ROM (fool proof) or on disk (most machines will have a disk or two anyhow).

companies, Microsoft has to be *the* major 'force' in micro software.

Bill Gates is the president of the company, and the APL job is largely his baby. So we asked him for his views on the micro APL scene.

NEAR FUTURE

What is about to happen in the way of micro APL?

'APL will see an incredible increase in popularity because it will be exposed to so many new people. To date, it has been an expensive language to use and is almost never introduced to first time computer users. APL's strengths assure that a significant percentage of personal computer users will adopt it as "the language". However it is not the ideal first time language and has some limitations of its own. BASIC will continue to dominate in this role, although specialized languages will be supported by personal computer manufacturers. Microsoft will introduce APL on the TRS-80, Exidy Sorcerer, Interact One, Nascom and NEC TK-80 in 1979.

Also the old time APL users will enjoy the low cost and ease of access to the low priced machines. These old

time users will be key in generating new converts. New tutorial material making APL seem less mathematical and not forcing the user to see the utility of all the operators will be required. The statistical, accounting, and model type applications that APL is excellent at will be key ones for the new small business computers. With APL's following, IBM's support of it and its strength, our OEMs have decided they can't ignore it. Our response on the product to date has been incredible.'

FIRST VERSION

Details of the 8080/Z80 version:

'Our 8080/Z80 version (long awaited) will be out in April running under CP/M. Other versions such as the virtual workspace one will follow.... Equivalence with the 5100 was my goal — and I met that with 24K of code.

We will license our APL to most of our existing OEMs.'

PERFORMANCE

What kind of performance is expected from micro APL?

'Microsoft APL has all the features of IBM APL/SV (360/370) APL except that the I/O and shared variables are handled differently and the math will not be totally the same since we use binary exponents where IBM uses hex. Our APL is twice as fast as the 5100 which is very adequate and often better than remote APL timesharing systems. Our 8086 and Z-8000 APLs will be at least five times faster and will be available in early 1980. This APL will outperform APL on most minis.

The string search and block transfer are an advantage that our Z-80 version will use. In specific cases this could cause a 2:1 speed difference, but overall the Z-80 to 8080 difference will be less than 15% at equal clock speeds. More important are the design techniques which allow a 32K byte work space to be equivalent to almost twice that much workspace on the 5100 through more efficient management and special representations.'

FAR FUTURE

Looking forward about a year Gates sees some of the following features being offered:

Workspaces upto 40K with virtual workspace to follow.

Libraries on disk.

Unlimited number of dimensions for arrays.

16 digit computation (floating point)

Availability of all primitives normally found.

Sophisticated system commands, error messages.

Initially only SAVE and LOAD workspace 'file' operations, but later random access I/O on arbitrary objects.

Later on Gates thinks 16 bit APL will have up to 10 meg workspaces, same library facilities as I.P. Sharp. Significantly, he comments that many language extensions may be persued, something he feels IBM has stifled. 'APL should not be allowed to stagnate. It is very weak in control structures. General lists should be added.'

AN EXCITING NEW COMPANY

Starting a new company can be a thrill, and it sounds like Microsoft is an example:

'Paul Allen and I started Microsoft in November 1974. The incredible potential for the microprocessor when combined with good software presented a unique opportunity for us to start on our own. The conventional languages and our enhancements of them (BASIC, FORTRAN and COBOL) have kept us extremely busy and delayed our introduction of APL for over 18 months. Providing these packages for the 16 bit processors as well as our new 8-bit products: a BASIC compiler, a Data base manager, a PASCAL based systems programming language and of course APL, will be our primary emphasis in 1979.

The APL project was started in January 1976. At one time people had doubted the ability of microprocessors to support high level languages. After BASIC had disproven this, APL seemed like a great follow-on product. APL provided a new challenge because of its complexity and requirement for lots of memory. , However, FORTRAN got higher priority and only one person was left on APL after November 1976. FORTRAN was introduced in June 1977. COBOL was introduced in June 1978.

Microsoft now employs 14 full time technical people and has moved to new and bigger offices in Seattle, Washington, as well as purchasing a DEC 2020 for in house development.'

THE MCM STORY

This company, based in Toronto and Kingston Ontaio, has had a very interesting past, and a unique part in the development of APL machines.

In late 1971, early 72, two fellows by the names of Mers Kutt and Gord Ramer started the company. Their first product was the MCM 70 desktop APL computer, which was based on the 8008 the first 8 bit microprocessor. This model was quickly superceded by the MCM 700 which was a slightly modified version. This machine had dual cassette drives, and one line alphanumeric display. 32K of ROM contained the interpreter, with 2K RAM provided, expandable to 8K.The 700 even came with battery power, enough for the APL addict to get his fix anywhere! This machine was offered for about $8000 in 1974.

Kutt left the company in mid 74, and in the spring of 75 Ted Berg became it's president.

In mid 76 the company's second model, the MCM 800 was introduced. Because the APL interpreter already developed was so good, but the 8008 a little slow, the 800 was centred on a CPU board built with a couple of ALUs and other supporting chips imitating the 8008. This resulted in a computation power improvement of around 10 over the 700. The 800 is generally used in a system with dual floppy disks, with perhaps a printer. About $9,000 could get you the 800 itself, which incorporated a VDU and single cassette, and came with 8K RAM.

In mid '78, Berg moved to take on MCM's US distributor Interactive Computer Systems Inc. The company's new president, Chuck Williams explains that with MCM's third generation machine, the MCM 900, the marketing strategy can now be more solidly aimed at a wide market of business applications. The 900 is based again on an 'imitation' high speed 8008, using a pair of 2901 4 bit slice processors, giving an increased performance of 5—10 over the 800 according to John Koiste, General Manager of MCM's Kingston R & D and production facility. Williams likens this improvement from 800 to 900 to the production of automobiles with top speeds of 60 and 80 miles an hour. While the performance improvement is relatively small, and the first car will accomplish transportation, there is a certain threshold level at which the product becomes widely acceptable. This, Williams feels, has happened with their 900, which strongly competes with timesharing and other more expensive models. The 900 itself is priced around $10,000, but is really intended for use with a dual disk and printer. Essentially it is part of an $18 — 20,000 business problem solving package.

As for the future, Koiste admits that MCM has gone about as far as it can with using the 8008 interpreter

ed) will be out in April running under CP/M. Other versions such as the virtual workspace one will follow.... Equivalence with the 5100 was my goal — and I met that with 24K of code.

We will license our APL to most of our existing OEMs.'

PERFORMANCE

What kind of performance is expected from micro APL?

'Microsoft APL has all the features of IBM APL/SV (360/370) APL except that the I/O and shared variables are handled differently and the math will not be totally the same since we use binary exponents where IBM uses hex. Our APL is twice as fast as the 5100 which is very adequate and often better than remote APL timesharing systems. Our 8086 and Z-8000 APLs will be at least five times faster and will be available in early 1980. This APL will outperform APL on most minis.

The string search and block transfer are an advantage that our Z-80 version will use. In specific cases this could cause a 2:1 speed difference, but overall the Z-80 to 8080 difference will be less than 15% at equal clock speeds. More important are the design techniques which allow a 32K byte work space to be equivalent to almost twice that much workspace on the 5100 through more efficient management and special representations.'

FAR FUTURE

Looking forward about a year

AN EXCITING NEW COMPANY

Starting a new company can be a thrill, and it sounds like Microsoft is an example:

'Paul Allen and I started Microsoft in November 1974. The incredible potential for the microprocessor when combined with good software presented a unique opportunity for us to start on our own. The conventional languages and our enhancements of them (BASIC, FORTRAN and COBOL) have kept us extremely busy and delayed our introduction of APL for over 18 months. Providing these packages for the 16 bit processors as well as our new 8-bit products: a BASIC compiler, a Data base manager, a PASCAL based systems programming language and of course APL, will be our primary emphasis in 1979.

The APL project was started in January 1976. At one time people had doubted the ability of microprocessors to support high level languages. After BASIC had disproven this, APL seemed like a great follow-on product. APL provided a new challenge because of its complexity and requirement for lots of memory. However, FORTRAN got higher priority and only one person was left on APL after November 1976. FORTRAN was introduced in June 1977. COBOL was introduced in June 1978.

Microsoft now employs 14 full time technical people and has moved to new and bigger offices in Seattle, Washington, as well as purchasing a DEC 2020 for in house development.'

In mid 76 the company's second model, the MCM 800 was introduced. Because the APL interpreter already developed was so good, but the 8008 a little slow, the 800 was centred on a CPU board built with a couple of ALUs and other supporting chips imitating the 8008. This resulted in a computation power improvement of around 10 over the 700. The 800 is generally used in a system with dual floppy disks, with perhaps a printer. About $9,000 could get you the 800 itself, which incorporated a VDU and single cassette, and came with 8K RAM.

In mid '78, Berg moved to take on MCM's US distributor Interactive Computer Systems Inc. The company's new president, Chuck Williams explains that with MCM's third generation machine, the MCM 900, the marketing strategy can now be more solidly aimed at a wide market of business applications. The 900 is based again on an 'imitation' high speed 8008, using a pair of 2901 4 bit slice processors, giving an increased performance of 5—10 over the 800 according to John Koiste, General Manager of MCM's Kingston R & D and production facility. Williams likens this improvement from 800 to 900 to the production of automobiles with top speeds of 60 and 80 miles an hour. While the performance improvement is relatively small, and the first car will accomplish transportation, there is a certain threshold level at which the product becomes widely acceptable. This, Williams feels, has happened with their 900, which strongly competes with time-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds $40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than $2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #114, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

Bill Gates

Bill Gates
General Partner, Micro-Soft

PhDs in 

Random Fact #1

RETROSPECTIVE THESES AND DISSERTATIONS

Idiom matching: an optimization technique for an APL compiler

Feng Sheng Cheng, *Iowa State University*

Degree Type
Dissertation

Date of Award
1981

Degree Name
Doctor of Philosophy

Department
Computer Science

Abstract
This thesis describes the design of an idiom matching technique in a compiler for APL. Idioms are programming language constructs used by programmers for the logical primitive operations for which no language primitives exist. Due to APL's richness of operators, idioms tend to appear at the "expression level." APL users have to pay a very high price to execute their APL programs with operator-by-operator execution in conventional translators. However, each idiom can be treated as a unit. It has been shown (2) that the saving from optimizing such idioms can be very impressive. Other researchers (1,3) have collected lists of idioms for APL;Idiom matching consists of two problems, recognition and selection, which are dealt with in this work. The idiom recognition problem is solved by a finite tree automaton. This approach constructs an automaton directly from a regular tree expression for a set of idioms. A practical automaton minimization algorithm is developed that obtains a time bound of O(n('2)(.)log n) for any binary tree automaton with n states. The selection problem, locating the best non-overlapping idioms in an expression tree, is solved in O(n) time;1. Brown, W. E. "Toward an Optimizing Compiler for a Very HighLevel Language." Ph.D. dissertation, Iowa State University, 1979;2. Miller, T. C. "Tentative Compilation: A Design for an APLCompiler." Ph.D. dissertation, Yale University, 1978;3. Perlis, A. J. and Rugaber, S. "The APL Idiom List." ResearchReport #87, Department of Computer Science, Yale University,April, 1977.

DOI
https://doi.org/10.31274/rtd-180813-4667

Idiom matching: an optimization technique for an APL compiler

dc.contributor.author	Cheng, Feng
dc.contributor.department	Computer Science
dc.date	2018-08-17T12:59:25.000
dc.date.accessioned	2020-07-02T05:57:10Z
dc.date.available	2020-07-02T05:57:10Z
dc.date.copyright	Thu Jan 01 00:00:00 UTC 1981
dc.date.issued	1981
dc.description.abstract	This thesis describes the design of an idiom matching technique in a compiler for APL. Idioms are programming language constructs used by programmers for the logical primitive operations for which no language primitives exist. Due to APL's richness of operators, idioms tend to appear at the "expression level." APL users have to pay a very high price to execute their APL programs with operator-by-operator execution in conventional translators. However, each idiom can be treated as a unit. It has been shown (2) that the saving from optimizing such idioms can be very impressive. Other researchers (1,3) have collected lists of idioms for APL;Idiom matching consists of two problems, recognition and selection, which are dealt with in this work. The idiom recognition problem is solved by a finite tree automaton. This approach constructs an automaton directly from a regular tree expression for a set of idioms. A practical automaton minimization algorithm is developed that obtains a time bound of O(n('2)(.)log n) for any binary tree automaton with n states. The selection problem, locating the best non-overlapping idioms in an expression tree, is solved in O(n) time;1. Brown, W. E. "Toward an Optimizing Compiler for a Very HighLevel Language." Ph.D. dissertation, Iowa State University, 1979;2. Miller, T. C. "Tentative Compilation: A Design for an APLCompiler." Ph.D. dissertation, Yale University, 1978;3. Perlis, A. J. and Rugaber, S. "The APL Idiom List." ResearchReport #87, Department of Computer Science, Yale University,April, 1977.

2020-07-02T05:57:10Z

Thu Jan 01 00:00:00 UTC 1981

1981

This thesis describes the design of an idiom matching technique in a compiler for APL. Idioms are programming language constructs used by programmers for the logical primitive operations for which no language primitives exist. Due to APL's richness of operators, idioms tend to appear at the "expression level." APL users have to pay a very high price to execute their APL programs with operator-by-operator execution in conventional translators. However, each idiom can be treated as a unit. It has been shown (2) that the saving from optimizing such idioms can be very impressive. Other researchers (1,3) have collected lists of idioms for APL;Idiom matching consists of two problems, recognition and selection, which are dealt with in this work. The idiom recognition problem is solved by a finite tree automaton. This approach constructs an automaton directly from a regular tree expression for a set of idioms. A practical automaton minimization algorithm is developed that obtains a time bound of O(n('2)(.)log n) for any binary tree automaton with n states. The selection problem, locating the best non-overlapping idioms in an expression tree, is solved in O(n) time;1. Brown, W. E. "Toward an Optimizing Compiler for a Very HighLevel Language." Ph.D. dissertation, Iowa State University, 1979;2. Miller, T. C. "Tentative Compilation: A Design for an APLCompiler." Ph.D. dissertation, Yale University, 1978;3. Perlis, A. J. and Rugaber, S. "The APL Idiom List." ResearchReport #87, Department of Computer Science, Yale University,April, 1977.

Toward an optimizing compiler for a very high level language

by

Walter E. Brown

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

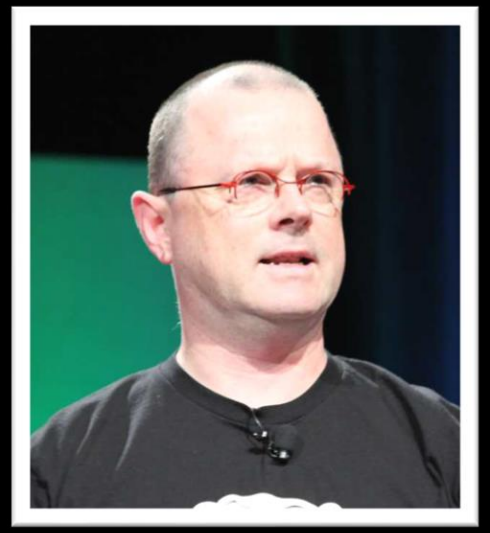
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

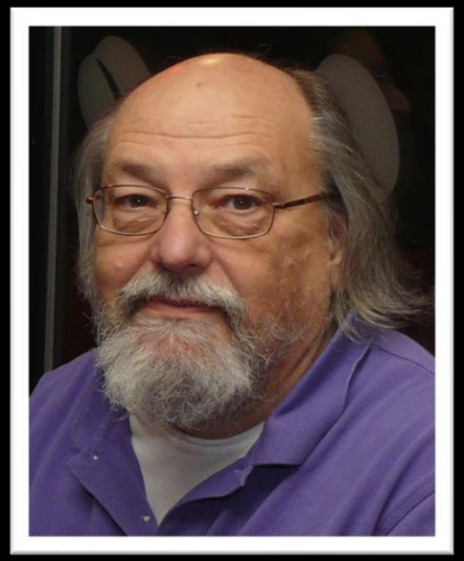
Random Fact #2

The Three Creators of



@rob_pike

@robertgriesemer



Ken Thompson
no twitter

No Raw Loops

Random Fact #3

No Raw Loops

www.NoRawLoops.com

www.norawloops.com

www.norawloops.com
www.nsl.com

www.norawloops.com
www.nsl.com
“No Stinking Loops”

no stinking loops

WOLF! (Andrew Chase)

- Wolf Construction Documentation

Tableau Systems (K3)

- Unification (Chang & Lee)
- KE Calculus for Truth-Functional Logic
- Modal Logics (K, T, D, K4, B, S4, S5) [incomplete - only K]
- Godel-Lob Provability Logic
- First-order Logic with Identity [incomplete]
- Truth-Functional Logic
- Trees as Vectors
- General Tree Diagrammer
- Binary Tree Diagrammer
- Nested Tree Diagrammer
- Iverson NV Parser

Negative Index

Random Fact #4

Negative Index

End of Mini Histories
& Random Facts

reduce

APL	**/ (reduce)**	-	Doc
CUDA	**reduce**	Thrust	Doc
D	**reduce**	algorithm.iteration	Doc
Ruby	**reduce**	Enumerable	Doc
Pharo	**reduce:**	Collection	Doc
Python	**reduce**	itertools	Doc
Elixir	**reduce**	Enum	Doc
Kotlin	**reduce**	collections	Doc
Scala	**reduce**	various	Doc
Clojure	**reduce**	core	Doc
C++	**reduce**	<numeric>	Doc
Rust	**fold**	trait.Iterator	Doc
Scala	**fold**	various	Doc
D	**fold**	algorithm.iteration	Doc
Pharo	**fold:**	Collection	Doc
Kotlin	**fold**	collections	Doc
BQN	**´ (fold)**		Doc
BQN	**˝ (insert)**	-	Doc
J	**/ (insert)**	-	Doc
Haskell	**foldl**	Data.List	Doc
Racket	**foldl**	base	Doc
q	**over**	-	Doc
Pharo	**inject:into:**	Collection	Doc
APL	**⌿ (reduce first)**		Doc
C#	**Aggregate**	Enumerable	Doc
Ruby	**inject**	Enumerable	Doc
C++	**accumulate**	<numeric>	Doc

	APL	/ (reduce)	-	Doc
	CUDA	reduce	Thrust	Doc
	D	reduce	algorithm.iteration	Doc
	Ruby	reduce	Enumerable	Doc
	Pharo	reduce:	Collection	Doc
	Python	reduce	itertools	Doc
	Elixir	reduce	Enum	Doc
	Kotlin	reduce	collections	Doc
	Scala	reduce	various	Doc
	Clojure	reduce	core	Doc
	C++	reduce	<numeric>	Doc
	Rust	fold	trait.Iterator	Doc
	Scala	fold	various	Doc
	D	fold	algorithm.iteration	Doc

	D	**fold**	algorithm.iteration	Doc
	Pharo	**fold:**	Collection	Doc
	Kotlin	**fold**	collections	Doc
	BQN	**´ (fold)**		Doc
	BQN	**˝ (insert)**	-	Doc
	J	**/ (insert)**	-	Doc
	Haskell	**foldl**	Data.List	Doc
	Racket	**foldl**	base	Doc
	q	**over**	-	Doc
	Pharo	**inject:into:**	Collection	Doc
	APL	**⌿ (reduce first)**		Doc
	C#	**Aggregate**	Enumerable	Doc
	Ruby	**inject**	Enumerable	Doc
	C++	**accumulate**	<numeric>	Doc

```cpp
// iota 5
auto a = std::vector<int>(5);
std::iota(a.begin(), a.end(), 1);
// + reduce iota 5
auto t = std::vector<int>(5);
std::iota(t.begin(), t.end(), 1);
auto b = std::accumulate(t.cbegin(), t.cend(), 0);
// + scan iota 5
auto c = std::vector<int>(5);
std::iota(c.begin(), c.end(), 1);
std::partial_sum(c.cbegin(), c.cend(), c.begin());

for (auto e : a) std::cout << e << ' ';
std::cout << '\n';
std::cout << b << '\n';
for (auto e : c) std::cout << e << ' ';
std::cout << '\n';
```

https://godbolt.org/z/jxa4jcPYh

```cpp
auto a = rv::iota(1, 6);
auto b = ranges::fold_left_first(rv::iota(1, 6), std::plus{}).value();
auto c = rv::iota(1, 6) | rv::partial_sum;

fmt::print("{}\n", a);  // [1, 2, 3, 4, 5]
fmt::print("{}\n", b);  // 15
fmt::print("{}\n", c);  // [1, 3, 6, 10, 15]
```

https://godbolt.org/z/74cdTrcde

```cpp
auto a = rv::iota(1, 6);
auto b = rv::iota(1, 6) |> ranges::fold_left_first(_, std::plus{}).value();
auto c = rv::iota(1, 6) |> ranges::scan_left_first(_, std::plus{}).value();

fmt::print("{}\n", a);  // [1, 2, 3, 4, 5]
fmt::print("{}\n", b);  // 15
fmt::print("{}\n", c);  // [1, 3, 6, 10, 15]
```

 98	 17	 23
accumulate partial_sum	reduce inclusive_scan exclusive_scan	fold_left fold_right fold_left_first fold_right_last

	Direction	Initial Value	Requires Associativity	Requires Commutativity	Range Overload
accumulate	*iterator*	✅	⛔	⛔	⛔
reduce	non-deterministic	✅	✅	✅	⛔
fold_left	left	✅	⛔	⛔	✅
fold_left_first	left	⛔	⛔	⛔	✅
fold_right	right	✅	⛔	⛔	✅
fold_right_last	right	⛔	⛔	⛔	✅


```
iota              ← ⍳
accumulate        ← +⌿
partial_sum       ← +\
inner_product     ← +.×
adjacent_difference ← 2-⍨⌿
```


```
iota                ← ⍳
accumulate          ← +/
partial_sum         ← +\
inner_product       ← +.×
adjacent_difference ← 2-⍨/

min_element     ← ⌊/
max_element     ← ⌈/
minmax_element  ← ⌊/,⌈/

any_of   ← {∨/⍺⍺⍵}
all_of   ← {∧/⍺⍺⍵}
none_of  ← {~∨/⍺⍺⍵}

count    ← {+/⍺=⍵}
count_if ← {+/⍺⍺⍵}

reverse ← ⌽
rotate  ← ⌽
```


```
any_of    ← { ∨/ααω}
all_of    ← { ∧/ααω}
none_of   ← {~∨/ααω}
count_if  ← { +/ααω}
```

```
_AnyOf    ← { ∨´𝕎𝕩}
_AllOf    ← { ∧´𝕎𝕩}
_NoneOf   ← {¬∨´𝕎𝕩}
_CountIf  ← { +´𝕎𝕩}
Count     ← { +´𝕨=𝕩}
```

scan

	Elixir	**scan**	Enum	Doc
	F#	**scan**	Seq	Doc
	APL	**\ (scan)**	-	Doc
	Scala	**scan**	various	Doc
	Rust	**scan**	trait.Iterator	Doc
	q	**scan**	-	Doc
	BQN	**` (scan)**	-	Doc
	C++	**inclusive_scan**	<numeric>	Doc
	CUDA	**inclusive_scan**	Thrust	Doc
	Kotlin	**runningReduce**	collections	Doc
	J	**\ (prefix)**	-	Doc
	Clojure	**reductions**	core	Doc
	Kotlin	**runningFold**	collections	Doc
	Python	**accumulate**	itertools	Doc
	Haskell	**scanl1**	Data.List	Doc
	C++	**partial_sum**	<numeric>	Doc
	D	**cumulativeFold**	algorithm.iteration	Doc


```
'<div>Hello <b>C++North!</b></div>'
```


```
'<'='<div>Hello <b>C++North!</b></div>'
```


```
'<'='<div>Hello <b>C++North!</b></div>'
```

```
1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
```


```
'<'='<div>Hello <b>C++North!</b></div>'
      1000000000010000000000010001000000
```


```
'<'='<div>Hello <b>C++North!</b></div>'
```

```
1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
```


```
'<'=
      '<div>Hello <b>C++North!</b></div>'

1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
```


```
      '<'=
'<div>Hello <b>C++North!</b></div>'

1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
```


```
{'<'=⍵}
'<div>Hello <b>C++North!</b></div>'

1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
```


```
{ω∈'<>'}
'<div>Hello <b>C++North!</b></div>'

1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1
```


```
{≠\⍵∊'<>'}
'<div>Hello <b>C++North!</b></div>'

1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 0
```


```
{~≠\⍵∈'<>'}
'<div>Hello <b>C++North!</b></div>'

0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1
```


```
{ω/⍨~≠\ω∈'<>'}
'<div>Hello <b>C++North!</b></div>'

0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1
```


```
{ω/⍨~≠\ω∈'<>'}
'<div>Hello <b>C++North!</b></div>'

< d i v > H e l l o   < b > C + + N o r t h ! < / b > < / d i v >
0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 1
```


```
{ω/⍨~≠\ω∈'<>'}
'<div>Hello <b>C++North!</b></div>'

>Hello >C++North!>>
```


```
{ω/⍨(~⊢∨≠\)ω∈'<>'}
'<div>Hello <b>C++North!</b></div>'
```

Hello C++North!


```
{ω/⍨(⊢⍨≠\)ω∊'<>'}
'<div>Hello <b>C++North!</b></div>'
```

Hello C++North!

```
auto filter_out_html_tags(std::string_view sv) {
    // ...
}
```

https://godbolt.org/z/qqKvj3jzx

```cpp
auto filter_out_html_tags(std::string_view sv) {
    auto angle_bracket_mask =
        sv | rv::transform([](auto e) { return e == '<' or e == '>'; });
    return // ...
}
```

https://godbolt.org/z/qqKvj3jzx

```cpp
auto filter_out_html_tags(std::string_view sv) {
    auto angle_bracket_mask =
        sv | rv::transform([](auto e) { return e == '<' or e == '>'; });
    return rv::zip(rv::zip_with(std::logical_or{},
            angle_bracket_mask,
            angle_bracket_mask | rv::partial_sum(std::not_equal_to{})), sv)
        // ...
}
```

https://godbolt.org/z/qqKvj3jzx

```cpp
auto filter_out_html_tags(std::string_view sv) {
    auto angle_bracket_mask =
        sv | rv::transform([](auto e) { return e == '<' or e == '>'; });
    return rv::zip(rv::zip_with(std::logical_or{},
            angle_bracket_mask,
            angle_bracket_mask | rv::partial_sum(std::not_equal_to{})), sv)
        | rv::filter([](auto t) { return not std::get<0>(t); })
        // ...
}
```

https://godbolt.org/z/qqKvj3jzx

```cpp
auto filter_out_html_tags(std::string_view sv) {
    auto angle_bracket_mask =
        sv | rv::transform([](auto e) { return e == '<' or e == '>'; });
    return rv::zip(rv::zip_with(std::logical_or{},
            angle_bracket_mask,
            angle_bracket_mask | rv::partial_sum(std::not_equal_to{})), sv)
        | rv::filter([](auto t) { return not std::get<0>(t); })
        | rv::transform([](auto t) { return std::get<1>(t); })
        // ...
}
```

https://godbolt.org/z/qqKvj3jzx

```cpp
auto filter_out_html_tags(std::string_view sv) {
    auto angle_bracket_mask =
        sv | rv::transform([](auto e) { return e == '<' or e == '>'; });
    return rv::zip(rv::zip_with(std::logical_or{},
            angle_bracket_mask,
            angle_bracket_mask | rv::partial_sum(std::not_equal_to{})), sv)
        | rv::filter([](auto t) { return not std::get<0>(t); })
        | rv::transform([](auto t) { return std::get<1>(t); })
        | ranges::to<std::string>;
}
```

https://godbolt.org/z/qqKvj3jzx

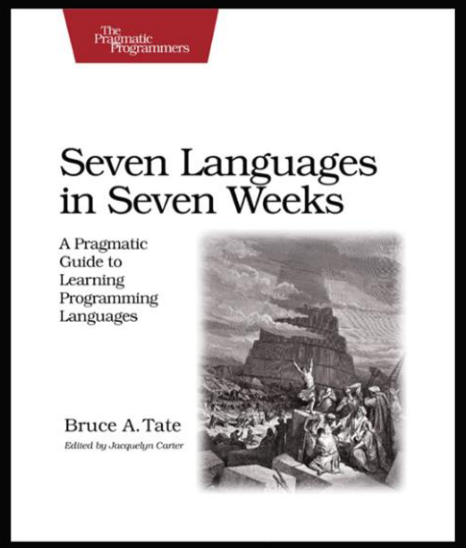
```
auto filter_out_html_tags(std::string_view sv) {
    return sv
         |  rv::transform(_phi(_eq('<'), _or_, _eq('>')))
         |> rv::zip_with(_or_, _, _ | rv::scan_left(_not_eq_))
         |> rv::zip(_, sv)
         |  rv::filter(_b(_not, _get<0>{}))
         |  rv::transform(_get<1>{})
         |  ranges::to<std::string>;
}
```

```
not
std::logical_not{}
```

outer_product

	Clojure	`outer-product`	`core.matrix`	Doc
	APL	`∘. (outer product)`	-	Doc
	Haskell	`outerProduct`	`Data.List.HT`	Doc
	BQN	`⌜ (table)`		Doc
	J	`/ (table)`		Doc
	R	`outer`	-	Doc

[[digression]]



7 Languages in 7 Weeks

Prolog

	Clojure	`outer-product`	`core.matrix`	Doc
	APL	`∘. (outer product)`	-	Doc
	Haskell	`outerProduct`	`Data.List.HT`	Doc
	BQN	`⌜ (table)`		Doc
	J	`/ (table)`		Doc
	R	`outer`	-	Doc

1. 
2. 

3. 
4. 
5. 

Combinator Introductions

	Lambda Expression	Bird Name	APL	BQN	Haskell	Other	Introduced
I	`λa.a`	Identity	Same	Identity	`id`		Sch24
K	`λab.a`	Kestrel	Right	Right	`const`		Sch24
KI	`λab.b`	Kite	Left	Left			
S	`λabc.ac(bc)`	Starling		After	`<*>`	Hook (J)	Sch24
B	`λabc.a(bc)`	Bluebird	Atop	Atop	`.`		Cur29
C	`λabc.acb`	Cardinal	Commute	Swap	`flip`	`SWAP` (FORTH)	Cur29
W	`λab.abb`	Warbler	Self(ie)	Self	`join`	`DUP` (FORTH)	Cur29
B1	`λabcd.a(bcd)`	Blackbird	Atop	Atop	`.:`		Cur58
Ψ	`λabcd.a(bc)(bd)`	Psi	Over	Over	`on`		Cur58
S'	`λabcd.a(bd)(cd)`	Phoenix	Fork	Fork	`liftA2`	Infix Notation (FP)	Tur79
E	`λabcde.ab(cde)`	Eagle					Smu85
Ê	`λabcdefg.a(bcd)(efg)`	Bald Eagle					Smu85
D2	`λabcde.a(bd)(ce)`	Dovekie		Before w/ After			Smu85
D	`λabcd.ab(cd)`	Dove	Beside	After			Smu85
🚫	`λabcde.a(bde)(cde)`	*Golden Eagle*	Fork	Fork			Iv89
🚫	`λabc.a(bc)c`	*Violet Starling*		Before		backHook (I)	Loc12
🚫	`λabcd.a(bc)d`	*Zebra Dove*		Before			
🚫	`λabcde.a(bcd)e`	*Harpy Eagle*					

Combinator Introductions

	Lambda Expression	Bird Name	APL	BQN	Haskell	Other	Introduced
I	`λa.a`	Identity	Same	Identity	`id`		Sch24
K	`λab.a`	Kestrel	Right	Right	`const`		Sch24
KI	`λab.b`	Kite	Left	Left			
S	`λabc.ac(bc)`	Starling		After	`<*>`	Hook (J)	Sch24
B	`λabc.a(bc)`	Bluebird	Atop	Atop	`.`		Sch24
C	`λabc.acb`	Cardinal	Commute	Swap	`flip`	`SWAP` (FORTH)	Sch24
W	`λab.abb`	Warbler	Self(ie)	Self	`join`	`DUP` (FORTH)	Cur29
B_1	`λabcd.a(bcd)`	Blackbird	Atop	Atop	`.:`		Cur30
Φ	`λabcd.a(bd)(cd)`	Phoenix	Fork	Fork	`liftA2`	Infix Notation (FP)	Cur31
$Φ_1$	`λabcde.a(bde)(cde)`	Pheasant	Fork	Fork			Cur31
Ψ	`λabcd.a(bc)(bd)`	Psi	Over	Over	`on`		Cur31
E	`λabcde.ab(cde)`	Eagle					Smu85
Ê	`λabcdefg.a(bcd)(efg)`	Bald Eagle					Smu85
D_2	`λabcde.a(bd)(ce)`	Dovekie		Before w/ After			Smu85
D	`λabcd.ab(cd)`	Dove	Beside	After			Smu85
🚫	`λabc.a(bc)c`	*Violet Starling*		Before		backHook (I)	Loc12
🚫	`λabcd.a(bc)d`	*Zebra Dove*		Before			
🚫	`λabcde.a(bcd)e`	*Harpy Eagle*					

Combinator Introductions

	Lambda Expression	Bird Name	APL	BQN	Haskell	Other	Introduced
I	`λa.a`	Identity	Same	Identity	`id`		Sch24
K	`λab.a`	Kestrel	Right	Right	`const`		Sch24
KI	`λab.b`	Kite	Left	Left			
S	`λabc.ac(bc)`	Starling		After	`<*>`	Hook (J)	Sch24
B	`λabc.a(bc)`	Bluebird	Atop	Atop	`.`		Sch24
C	`λabc.acb`	Cardinal	Commute	Swap	`flip`	`SWAP` (FORTH)	Sch24
W	`λab.abb`	Warbler	Self(ie)	Self	`join`	`DUP` (FORTH)	Cur29
B_1	`λabcd.a(bcd)`	Blackbird	Atop	Atop	`.:`		Cur30
Φ	`λabcd.a(bd)(cd)`	Phoenix	Fork	Fork	`liftA2`	Infix Notation (FP)	Cur31
$Φ_1$	`λabcde.a(bde)(cde)`	Pheasant	Fork	Fork			Cur31
Ψ	`λabcd.a(bc)(bd)`	Psi	Over	Over	`on`		Cur31
E	`λabcde.ab(cde)`	Eagle					Smu85
Ê	`λabcdefg.a(bcd)(efg)`	Bald Eagle					Smu85
D_2	`λabcde.a(bd)(ce)`	Dovekie		Before w/ After			Smu85
D	`λabcd.ab(cd)`	Dove	Beside	After			Smu85
🚫	`λabc.a(bc)c`	*Violet Starling*		Before		backHook (I)	Loc12
🚫	`λabcd.a(bc)d`	*Zebra Dove*		Before			
🚫	`λabcde.a(bcd)e`	*Harpy Eagle*					

Combinator Introductions

	Lambda Expression	Bird Name	APL	BQN	Haskell	Other	Introduced
I	`λa.a`	Identity	Same	Identity	`id`		Sch24
K	`λab.a`	Kestrel	Right	Right	`const`		Sch24
KI	`λab.b`	Kite	Left	Left			
S	`λabc.ac(bc)`	Starling		After	`<*>`	Hook (J)	Sch24
B	`λabc.a(bc)`	Bluebird	Atop	Atop	`.`		Sch24
C	`λabc.acb`	Cardinal	Commute	Swap	`flip`	`SWAP` (FORTH)	Sch24
W	`λab.abb`	Warbler	Self(ie)	Self	`join`	`DUP` (FORTH)	Cur29
B_1	`λabcd.a(bcd)`	Blackbird	Atop	Atop	`.:`		Cur30
Φ	`λabcd.a(bd)(cd)`	Phoenix	Fork	Fork	`liftA2`	Infix Notation (FP)	Cur31
$Φ_1$	`λabcde.a(bde)(cde)`	Pheasant	Fork	Fork			Cur31
Ψ	`λabcd.a(bc)(bd)`	Psi	Over	Over	`on`		Cur31
E	`λabcde.ab(cde)`	Eagle					Smu85
Ê	`λabcdefg.a(bcd)(efg)`	Bald Eagle					Smu85
D_2	`λabcde.a(bd)(ce)`	Dovekie		Before w/ After			Smu85
D	`λabcd.ab(cd)`	Dove	Beside	After			Smu85
🚫	`λabc.a(bc)c`	*Violet Starling*		Before		backHook (I)	Loc12
🚫	`λabcd.a(bc)d`	*Zebra Dove*		Before			
🚫	`λabcde.a(bcd)e`	*Harpy Eagle*					

[[end of digression]]

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

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

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

B. Snow Walking Robot

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Recently you have bought a snow walking robot and brought it home. Suppose your home is a cell $(0, 0)$ on an infinite grid.

You also have the sequence of instructions of this robot. It is written as the string s consisting of characters 'L', 'R', 'U' and 'D'. If the robot is in the cell (x, y) right now, he can move to one of the adjacent cells (depending on the current instruction).

- If the current instruction is 'L', then the robot can move to the left to $(x - 1, y)$;
- if the current instruction is 'R', then the robot can move to the right to $(x + 1, y)$;
- if the current instruction is 'U', then the robot can move to the top to $(x, y + 1)$;
- if the current instruction is 'D', then the robot can move to the bottom to $(x, y - 1)$.

You've noticed the warning on the last page of the manual: if the robot visits some cell (**except** $(0, 0)$) twice then it breaks.

So the sequence of instructions is valid if the robot starts in the cell $(0, 0)$, performs the given instructions, visits no cell other than $(0, 0)$ two or more times and ends the path in the cell $(0, 0)$. Also cell $(0, 0)$ should be visited **at most** two times: at the beginning and at the end (if the path is empty then it is visited only once). For example, the following sequences of instructions are considered valid: "UD", "RL", "UUURULLDDDDLDDRRUU", and the following are considered invalid: "U" (the endpoint is not $(0, 0)$) and "UUDD" (the cell $(0, 1)$ is visited twice).

The initial sequence of instructions, however, might be not valid. You don't want your robot to break so you decided to reprogram it in the following way: you will remove some (possibly, all or none) instructions from the initial sequence of instructions, then rearrange the remaining instructions as you wish and turn on your robot to move.

Your task is to remove as few instructions from the initial sequence as possible and rearrange the remaining ones so that the sequence is valid. Report the valid sequence of the maximum length you can obtain.

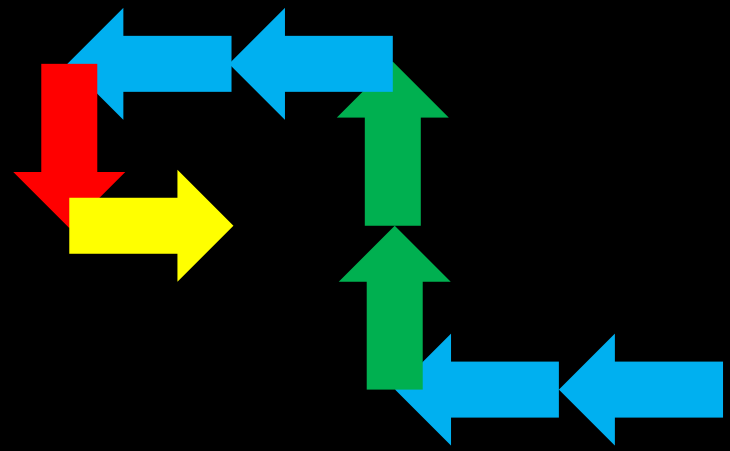
Note that you can choose **any** order of remaining instructions (you don't need to minimize the number of swaps or any other similar metric).

You have to answer q independent test cases.

http://codeforces.com/contest/1272/problem/B

L L U U L L D R

LLUULLDR



```cpp
auto solve(string s) -> string {
    auto u  = count(s.begin(), s.end(), 'U');
    auto d  = count(s.begin(), s.end(), 'D');
    auto l  = count(s.begin(), s.end(), 'L');
    auto r  = count(s.begin(), s.end(), 'R');
    auto ud = min(u, d);
    auto lr = min(l, r);
    if (lr == 0 && ud == 0) return "";
    if (ud == 0) return "LR";
    if (lr == 0) return "UD";
    string ans(ud,           'U');
    ans.resize(ud + lr,      'L');
    ans.resize(ud*2 + lr,    'D');
    ans.resize(ud*2 + lr*2, 'R');
    return ans;
}
```

https://codeforces.com/contest/1272/submission/66690239

```cpp
auto solve(string s) -> string {
    auto u  = count(s.begin(), s.end(), 'U');
    auto d  = count(s.begin(), s.end(), 'D');
    auto l  = count(s.begin(), s.end(), 'L');
    auto r  = count(s.begin(), s.end(), 'R');
    // ...
}
```

https://codeforces.com/contest/1272/submission/66690239

```cpp
auto solve(string s) -> string {
    auto u  = count(s.begin(), s.end(), 'U');
    auto d  = count(s.begin(), s.end(), 'D');
    auto l  = count(s.begin(), s.end(), 'L');
    auto r  = count(s.begin(), s.end(), 'R');
    // ...
}
```

https://codeforces.com/contest/1272/submission/66690239


```
'LRDLURDLURDULR'
```


```
'LRUD'∘.='LRDLURDLURDULR'
```


```
'LRUD'∘.='LRDLURDLURDULR'
```

```
1 0 0 1 0 0 0 1 0 0 0 0 1 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 0 1 0 0 1 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0
```


```
'LRUD'∘.,'LRDLURDLURDULR'
```

LL	LR	LD	LL	LU	LR	LD	LL	LU	LR	LD	LU	LL	LR
RL	RR	RD	RL	RU	RR	RD	RL	RU	RR	RD	RU	RL	RR
UL	UR	UD	UL	UU	UR	UD	UL	UU	UR	UD	UU	UL	UR
DL	DR	DD	DL	DU	DR	DD	DL	DU	DR	DD	DU	DL	DR


```
'LRDLURDLURDULR'
'LRUD'
```


```
      'LRDLURDLURDULR'
L
R
U
D
```


```
'LRUD'∘.='LRDLURDLURDULR'
```

```
1 0 0 1 0 0 0 1 0 0 0 0 1 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 0 1 0 0 1 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0
```


```
+/'LRUD'∘.='LRDLURDLURDULR'
```

```
1 0 0 1 0 0 0 1 0 0 0 0 1 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 1 0 0 0 1 0 0 1 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0
```


```
+/'LRUD'∘.='LRDLURDLURDULR'
```

```
4 4 3 3
```

```
auto outer_product(auto left, auto right, auto binop) {
    return rv::cartesian_product(left, right)
         | rv::transform([=](auto t) { return std::apply(binop, t); })
         | rv::chunk(ranges::distance(right));
}
```

https://godbolt.org/z/jEYvh9hYv

```cpp
auto outer_product(auto left, auto right, auto binop) {
    return rv::cartesian_product(left, right)
         | rv::transform([=](auto t) { return std::apply(binop, t); })
         | rv::chunk(ranges::distance(right));
}

auto count_lrud(std::string_view sv) {
    return outer_product("LRUD"sv, sv, std::equal_to{})
        | rv::transform([](auto r) {
            return ranges::fold_left_first(r, std::plus{}).value(); });
}
```

https://godbolt.org/z/jEYvh9hYv

```
auto outer_product(auto binop) {
    return [=](auto left, auto right) {
        return rv::cartesian_product(left, right)
             | rv::transform([=](auto t) { return std::apply(binop, t); })
             | rv::chunk(ranges::distance(right));
    };
}

auto count_lrud(std::string_view sv) {
    return outer_product(std::equal_to{})("LRUD"sv, sv)
        | rv::transform([](auto r) {
            return ranges::fold_left_first(r, std::plus{}).value(); });
}
```

https://godbolt.org/z/jEYvh9hYv

```python
def count_lrud(s):
    return list(Counter(s).values())
```

https://godbolt.org/z/TfdGdb61G

```haskell
import Data.List.HT (outerProduct)

countLRUD = map (sum . map fromEnum)
          . outerProduct (==) "LRUD"
```

```haskell
import Data.List.Unique (count)

countLRUD = map snd . count
```


```
countLRUD ← {+/'LRUD'∘.=⍵}
```

```
def count_lrud(s): return list(Counter(s).values())
```

```
countLRUD = map (sum . map fromEnum) . outerProduct (==) "LRUD"
countLRUD = map snd . count
```

```
auto count_lrud(std::string_view sv) {
    return outer_product(std::equal_to{})("LRUD"sv, sv)
        | rv::transform([](auto r) {
            return ranges::fold_left_first(r, std::plus{}).value(); });
}
```


```
countLRUD ← {+/'LRUD'∘.=⍵}
```

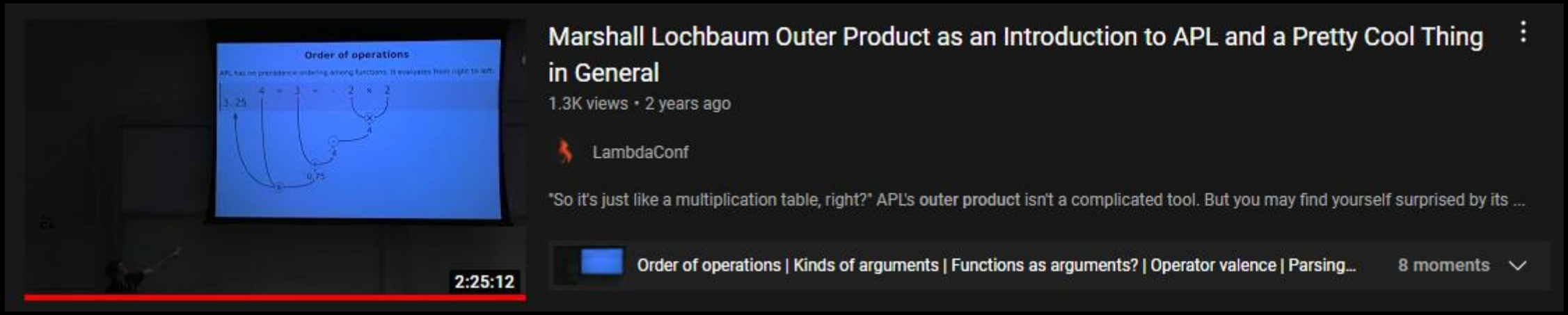

```
auto count_lrud(std::string_view sv) {
    return outer_product(std::equal_to{})("LRUD"sv, sv)
        | rv::transform([](auto r) {
            return ranges::fold_left_first(r, std::plus{}).value(); });
}
```


```
countLRUD ← {+/'LRUD'∘.=⍵}
```


```
auto count_lrud(std::string_view sv) {
    return outer_product(std::equal_to{})("LRUD"sv, sv)
        | rv::transform([](auto r) {
            return ranges::fold_left_first(r, std::plus{}).value(); });
}
```


```
countLRUD ← {+/'LRUD'∘.=⍵}
```


```
auto count_lrud(std::string_view sv) {
    return outer_product(std::equal_to{})("LRUD"sv, sv)
        |> rp::fold_left_first(_, std::plus{});
        |> values;
}
```



https://www.youtube.com/watch?v=WlUHw4hC4OY

rotate

& other beautiful glyphs


```
iota                ← ⍳
accumulate          ← +/
partial_sum         ← +\
inner_product       ← +.×
adjacent_difference ← 2-⍨/

min_element     ← ⌊/
max_element     ← ⌈/
minmax_element  ← ⌊/,⌈/

any_of   ← {∨/⍺⍺⍵}
all_of   ← {∧/⍺⍺⍵}
none_of  ← {~∨/⍺⍺⍵}

count    ← {+/⍺=⍵}
count_if ← {+/⍺⍺⍵}

reverse ← ⌽
rotate  ← ⌽
```

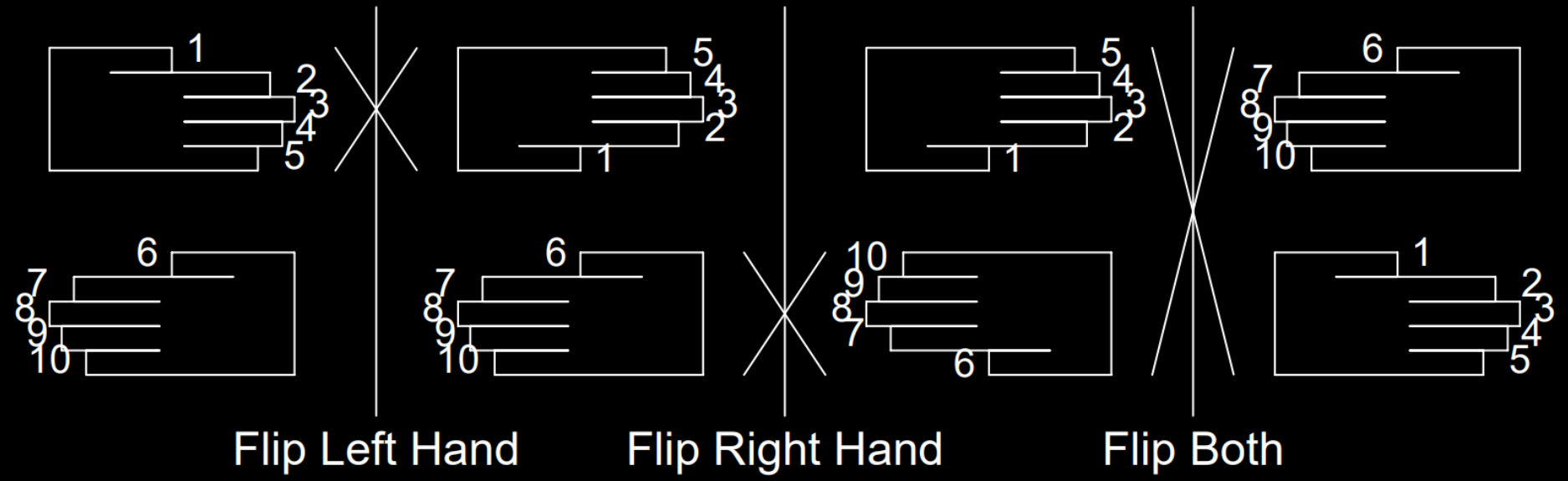

```
reverse ← ⌽
rotate  ← ⌽
```

```
void rotate(auto f, auto m, auto l) {
    std::reverse(f, m);
    std::reverse(m, l);
    std::reverse(f, l);
}
```

https://godbolt.org/z/vK6GGbefc

```
5216
5217  template <class _FwdIt>
5218  _CONSTEXPR20 _FwdIt rotate(_FwdIt _First, _FwdIt _Mid, _FwdIt _Last) {
5219      // exchange the ranges [_First, _Mid) and [_Mid, _Last)
5220      // that is, rotates [_First, _Last) left by distance(_First, _Mid) positions
5221      // returns the iterator pointing at *_First's new home
5222      _Adl_verify_range(_First, _Mid);
5223      _Adl_verify_range(_Mid, _Last);
5224      auto _UFirst      = _Get_unwrapped(_First);
5225      auto _UMid        = _Get_unwrapped(_Mid);
5226      const auto _ULast = _Get_unwrapped(_Last);
5227      if (_UFirst == _UMid) {
5228          return _Last;
5229      }
5230
5231      if (_UMid == _ULast) {
5232          return _First;
5233      }
5234
5235      if constexpr (_Is_random_iter_v<_FwdIt>) {
5236          _STD reverse(_UFirst, _UMid);
5237          _STD reverse(_UMid, _ULast);
5238          _STD reverse(_UFirst, _ULast);
5239          _Seek_wrapped(_First, _UFirst + (_ULast - _UMid));
5240      } else if constexpr (_Is_bidi_iter_v<_FwdIt>) {
5241          _STD reverse(_UFirst, _UMid);
5242          _STD reverse(_UMid, _ULast);
5243          auto _Tmp = _Reverse_until_sentinel_unchecked(_UFirst, _UMid, _ULast);
5244          _STD reverse(_Tmp.first, _Tmp.second);
5245          _Seek_wrapped(_First, _UMid != _Tmp.first ? _Tmp.first : _Tmp.second);
5246      } else {
5247          auto _UNext = _UMid;
5248          do { // rotate the first cycle
5249              _STD iter_swap(_UFirst, _UNext);
5250              ++_UFirst;
5251              ++_UNext;
5252              if (_UFirst == _UMid) {
5253                  _UMid = _UNext;
5254              }
5255          } while (_UNext != _ULast);
5256          _Seek_wrapped(_First, _UFirst);
5257          while (_UMid != _ULast) { // rotate subsequent cycles
5258              _UNext = _UMid;
5259              do {
5260                  _STD iter_swap(_UFirst, _UNext);
5261                  ++_UFirst;
5262                  ++_UNext;
5263                  if (_UFirst == _UMid) {
5264                      _UMid = _UNext;
5265                  }
5266              } while (_UNext != _ULast);
5267          }
5268      }
5269
5270      return _First;
5271  }
```

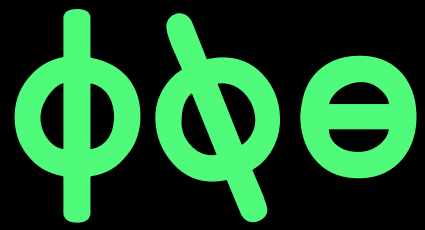
```
5235        if constexpr (_Is_random_iter_v<_FwdIt>) {
5236            _STD reverse(_UFirst, _UMid);
5237            _STD reverse(_UMid, _ULast);
5238            _STD reverse(_UFirst, _ULast);
5239            _Seek_wrapped(_First, _UFirst + (_ULast - _UMid));
5240        } else if constexpr (_Is_bidi_iter_v<_FwdIt>) {
5241            _STD reverse(_UFirst, _UMid);
5242            _STD reverse(_UMid, _ULast);
5243            auto _Tmp = _Reverse_until_sentinel_unchecked(_UFirst, _UMid, _ULast);
5244            _STD reverse(_Tmp.first, _Tmp.second);
5245            _Seek_wrapped(_First, _UMid != _Tmp.first ? _Tmp.first : _Tmp.second);
5246        } else {
```

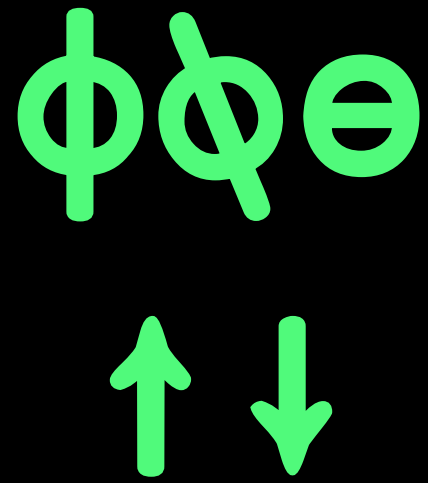

1
2
3
4
5
5
4
3
2
1
5
4
3
2
1
6
7
8
9
10
6
7
8
9
10
6
7
8
9
10
10
9
8
7
6
1
2
3
4
5
Flip Left Hand
Flip Right Hand
Flip Both

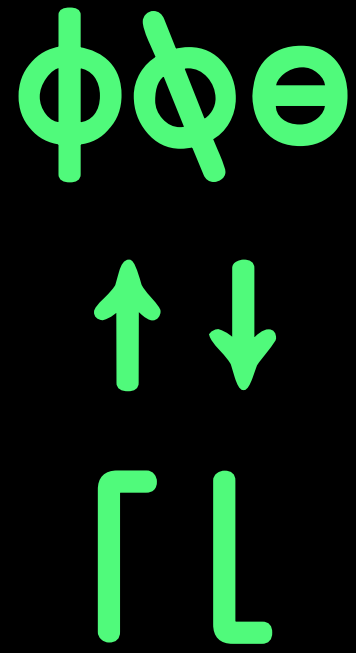
Programming Pearls

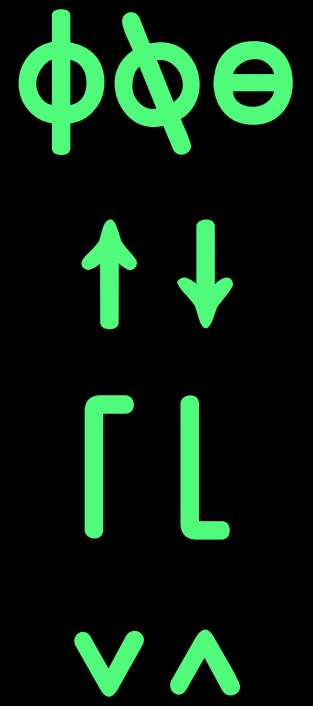
Second Edition

Jon Bentley







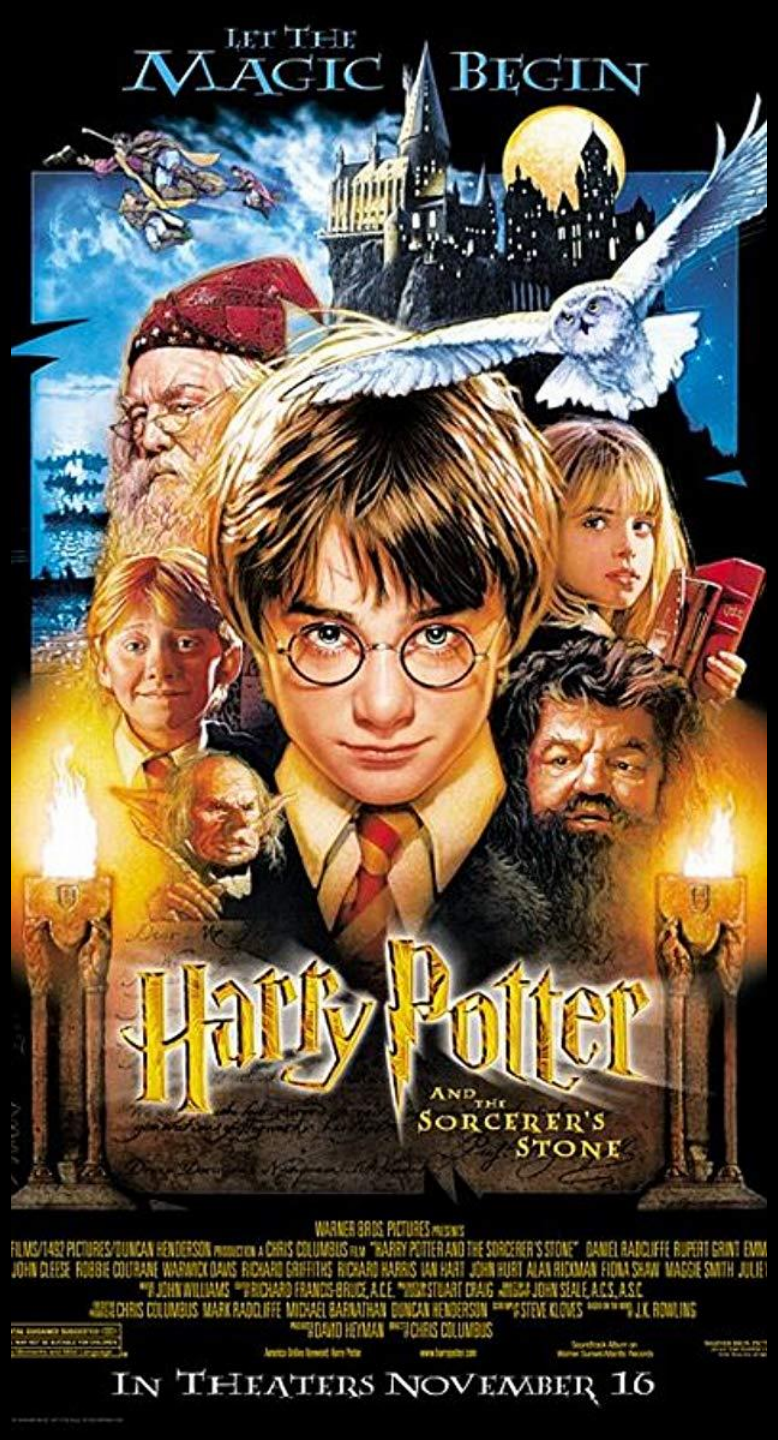


Harry Potter

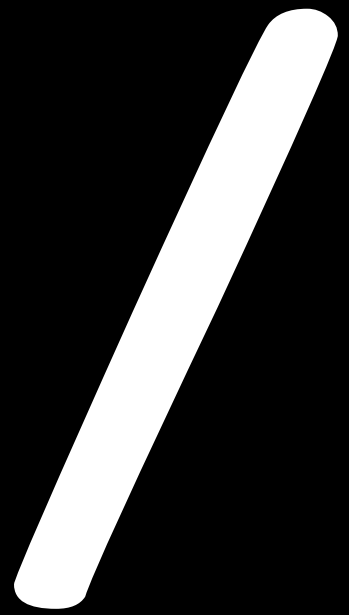
the Twin Algorithms

Conor Hoekstra

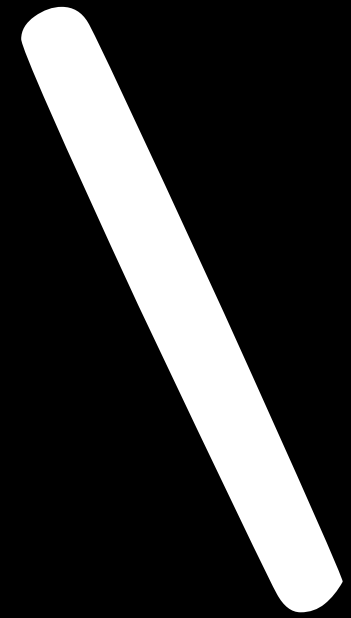
 code_report 


LET THE
MAGIC BEGIN
Harry Potter
AND THE
SORCERER'S
STONE
WARNER BROS. PICTURES PRESENTS
IN THEATERS NOVEMBER 16


EVERYTHING IS ABOUT TO CHANGE.
Harry Potter
AND THE GOBLET OF FIRE
THE ADVENTURE CONTINUES NOVEMBER 18



reduce



scan

In Summary

This talk wasn't about
convincing you to learn **APL**.

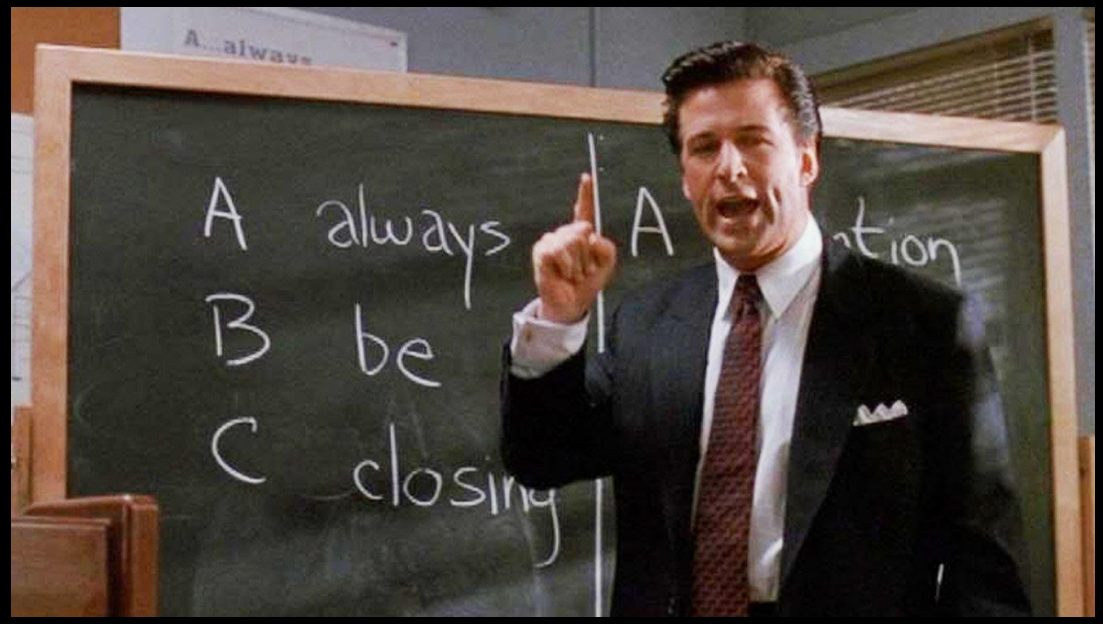
This talk was about convincing you to…

… be curious.
… think different.

… be curious.

… think different.

… **ABL** (**A**lways **B**e **L**earning).






"You can never understand **one** language until you understand at least **two**."

Geoffrey Williams





APL
















Rabbit Hole:

Who	What	Date	Link
Dave Thomas	Video	2015	Living in Big Data with Vector Functional Programming
Bernard Legrand	Article	2006	APL – a Glimpse of Heaven
Alan J. Perlis	Paper	1977	In praise of APL: a language for lyrical programming
---			
Richard Park	Video	2022	Easy to Learn - Worth Mastering
Rodrigo Serrão	Video	2022	Why APL is a language worth knowing
Conor Hoekstra	Video	2021	Algorithms as a Tool of Thought

Marshall Lochbaum	Video	2020	Outer Product as an Introduction to APL and a Pretty Cool Thing in General
---			
CoRecursive	Podcast	2021	Episode 65 - From Competitive Programming to APL
ArrayCast	Podcast	2021-Present	ArrayCast

https://github.com/codereport/Content/blob/main/Talks/2022-07-CppNorth/TheTwinAlgorithms/IfYouWantToContinueDownTheRabbitHole.md

Podcast Links:

Podcast	Guest	Date	Link
CppCast	Colin Hirsch	2018-08-09	Episode 162: The Art of C++ Libraries
CoRecursive	Conor Hoekstra	2021-06-02	Episode 64: From Competitive Programming to APL

YouTube Video Links:

Speaker	Conference/Meetup	Year	Talk
Alexander Stepanov	Talk at Adobe	2002	★ STL and Its Design Principles
Giovanni van Bronckhorst	FIFA World Cup	2010	Goal vs Uruguay
Sean Parent	Going Native	2013	C++ Seasoning
Conor Hoekstra	C++Now	2019	Algorithm Intuition
Conor Hoekstra	Meeting C++	2019	Better Algorithm Intuition
Conor Hoekstra	YouTube	2020	The STL Algorithm Cheat Sheet
Marshall Lochbaum	LambdaConf	2020	Outer Product as an Introduction to APL and a Pretty Cool Thing in General

Paper Links:

Author	Date	Link
Alan J. Perlis	1977	★ In praise of APL: a language for lyrical programming
Ken Iverson	1979	Notation as a Tool of Thought
Walter E Brown	1979	Toward an optimizing compiler for a very high level language
Feng Cheng	1981	Idiom matching: an optimization technique for an APL compiler

Article/Other Links:

Author	Site	Date	Link
Bernard Legrand	Vector	2006-06-22	★ APL – a Glimpse of Heaven
algoshift	HackerNews	2010-12-15	What's so cool about APL? (comment)
Scott Locklin	Locklin on Science	2013-07-13	Ruins of forgotten empires: APL languages
Noelkd	HackerNews	2014-05-28	APL – a Glimpse of Heaven (2006) (vector.org.uk)
Conor Hoekstra	GitHub	2022-03-12	Combinator Introductions

★ - Quoted

https://github.com/codereport/Content/blob/main/Talks/2022-07-CppNorth/TheTwinAlgorithms/Links.md

Thank you!

https://github.com/codereport/Content

Conor Hoekstra

code_report

codereport

Questions?

https://github.com/codereport/Content

Conor Hoekstra

code_report

codereport


```
count 5
ACCUMULATE plus count 5
REDUCE plus count 5
```