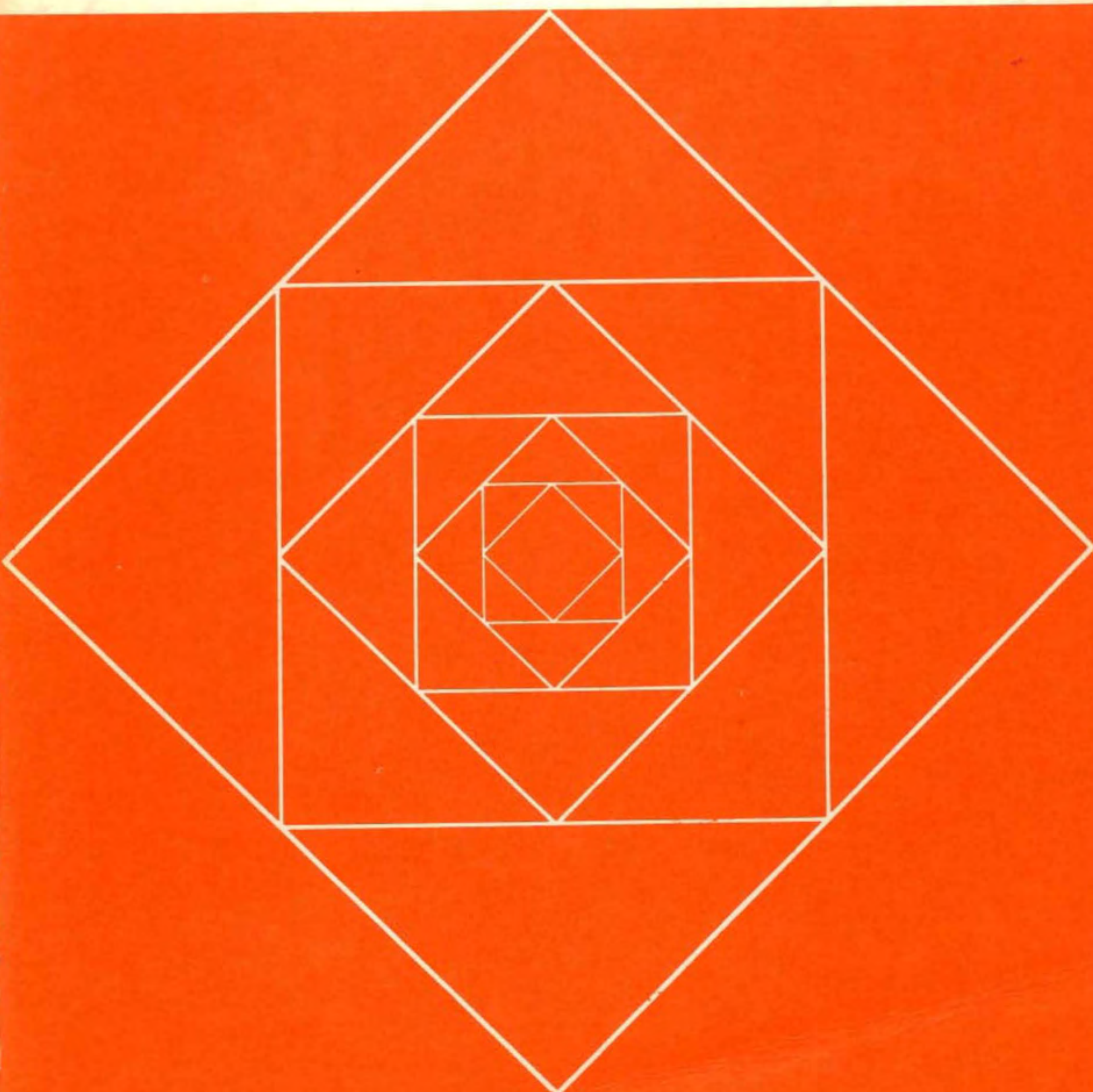
APL IN EXPOSITION

IBM PHILADELPHIA SCIENTIFIC CENTER
Data Processing Division

TECH. REPORT NO. 320-3010 JANUARY 1972

K. E. IVERSON



IBM PHILADELPHIA SCIENTIFIC CENTER TECHNICAL REPORT NO.320-3010 JANUARY 1972

APL IN EXPOSITION

K. E. IVERSON

PHILADELPHIA SCIENTIFIC CENTER

IBM CORP.

3401 MARKET STREET. PHILADELPHIA, PENNSYLVANIA 19104

THE USE OF APL IN EXPOSITION

The following pages illustrate the use of APL for exposition in the teaching of various topics. The first section presents the characteristics of the language, and each of the succeeding sections illustrates its use in the presentation of material in some one discipline.

A reader who wishes to study these examples thoroughly must either know the meaning of the APL notation used or be prepared to obtain this knowledge in some way, perhaps by inferring it from the examples, by consulting an APL manual, by experimenting on an APL terminal, or by asking a few questions of a native speaker of APL.

The treatment of each topic is self-contained, and so brief that it can only suggest the convenience provided by APL in more extended discussion. A perusal of several topics will illustrate the fact that the convenience of APL is not confined to any particular field. More extended use of the language is illustrated by some of the items in the bibliography.

This paper arose from material developed for a series of talks given at various locations over the past year or so. Its form betrays this origin; each page is relatively self-contained and is suitable for use as a transparency on an overhead projector. The following topics are treated:

I am greatly indebted to my colleagues at the Philadelphia Scientific Center, particularly to Messrs. Berry and Falkoff for suggestions on the treatment of computers, and to Mr. E. E. McDonnell for a critical reading of the manuscript.

APL\360

IS LIKE HIGH SCHOOL ALGEBRA:

In expressing familiar arithmetic functions.

```
      3.6+4.4
8
      3.6×4.4
15.84
       3÷4
0.75
```

In using the same *form* to express less familiar functions.

```
      3*4
81
```

3 to the power 4.

```
      3|8
2
```

The remainder on dividing 3 into 8.

```
      3⌈8
8
```

The maximum of 3 and 8.

```
      3≤8
1
      8≤3
0
```

The truth (1) or falsity (0) of a relation.

```
      (8+3)×(8-3)
55
      (3⌈8)+(3≤8)
9
```

In using parentheses to indicate the sequence in which parts of an expression are to be executed.

BUT DIFFERS FROM ALGEBRA IN RESPECTS WHICH BOTH SIMPLIFY IT AND EXTEND ITS APPLICABILITY:

```
      X←3+4
      Y←5
      X×Y
35
```

A value is assigned to a name (variable) by the assign symbol ← rather than by the equal sign. This avoids the multiple uses of equal encountered in algebra.

```
      LENGTH←5
      WIDTH←4
      AREA←LENGTH×WIDTH
      AREA
20
```

The multiplication sign (×) cannot be omitted. This allows the use of long names (e.g., *AREA* is a name and does not mean *A×R×E×A*).

```
      PRICE←5
      QUANTITY←4
      PRICE×QUANTITY
20
      PRICE←5 8 12 3 7
      QUANTITY←4 1 0 2 2
      PRICE×QUANTITY
20 8 0 6 14

      NEWPRICE←6 7 12 4 8
      PRICE ⌊ NEWPRICE
5 7 12 3 7
```

Expressions apply to lists of items (vectors) as well as to single quantities (scalars).

```
      +/QUANTITY
9
      4+1+0+2+2
9
      TOTAL←+/PRICE×QUANTITY
      TOTAL
48
      ⌈/QUANTITY
4
      4⌈1⌈0⌈2⌈2
4
```

Any function can be applied to all elements of a list. In algebra this can be done for addition by using the sigma notation.

```
      4+5×6
34
      4×5+6
44
```

There are no rules such as "multiplication is done before addition"; all functions are treated alike by one rule: evaluate from right to left, subject to parentheses.

APL CONTAINS A RICH SET OF PRIMITIVE (I.E., BUILT-IN) FUNCTIONS WHICH MAKE IT APPLICABLE OVER A WIDE AREA. IT INCLUDES, FOR EXAMPLE:

* All common arithmetic functions, including remainder, integer part, and power.

* Other mathematical functions such as trigonometric and hyperbolic functions (and their inverses), the gamma function, matrix inverse, and generalized matrix products.

* Simple but powerful selection functions which select parts of lists or tables. These include indexing in which the indices may themselves be lists or tables. Since lists and tables of characters are treated in the same way as lists and tables of numbers, these functions make APL easy to use in textual and other non-numeric work.

* A complete set of relations and other logical functions.

NEVERTHELESS, APL IS EASY TO LEARN BECAUSE IT IS <u>SEPARABLE</u>, I.E.,

* In attacking a given problem area only the necessary primitives must be learned and the rest may be ignored.

* When one adds new functions to his vocabulary in order to attack new areas, the same familiar rules apply to these new functions.

APL IS CONVENIENT TO USE IN ANY APPLICATION AREA BECAUSE THE FUNCTIONS NEEDED TO TREAT THAT AREA CAN BE DEFINED AND THEN USED AS CONVENIENTLY AS PRIMITIVES. FOR EXAMPLE:

```
      ∇Z←RATE FOR YEARS
[1]   Z←⌊.5+1000×(1+.01×RATE)*YEARS∇

      6 FOR 1
1060
      6 FOR 2
1124
      6 FOR 3
1191
```

The function *FOR* defined to the left applies to any rate (in percent) and any number of years and yields the rounded return in dollars for each 1000 dollars of initial capital.

```
      6 FOR 1 2 3 4
1060  1124  1191  1262
```

It applies for any list of years at a given rate.

```
      6 7 8 9 FOR 4
1262  1311  1360  1412
```

It applies for any list of rates for a given number of years.

```
      6 7 8 9 FOR 1 2 3 4
1060  1145  1260  1412
```

Or to any list of corresponding rates and years.

```
      ∇Z←RATE FORTABLE YEARS
[1]   Z←⌊0.5+1000×(1+0.01×RATE)∘.*YEARS∇

      6 7 8 9 FORTABLE 1 2 3 4
   1060  1124  1191  1262
   1070  1145  1225  1311
   1080  1166  1260  1360
   1090  1188  1295  1412
```

A slight modification of the expression used in defining *FOR* yields a function which produces a table which includes the result for every combination of rates and years.

FURTHER DETAILS OF APL NEEDED TO READ THE REST OF THIS PAPER ARE SUMMARIZED ON THIS PAGE AND IN THE TWO SUCCEEDING TABLES (WHICH DEFINE ALL THE PRIMITIVE FUNCTIONS):

Functions apply to arrays in four distinct ways, defined below by examples using the following arrays:

```
          V                             M
1   2   3   4                       1   2   3
          W                         4   5   6
4   3   2   1                       7   8   9
```

Element-by-element

```
          V×W              M×M              M*2
4   6   6   4          1    4    9      1    4    9
          2×W         16   25   36     16   25   36
8   6   4   2         49   64   81     49   64   81
```

Outer-Product (All Pairs)

```
     V∘.≤W              V∘.×W                  V∘.+W
1 1 1 1            4    3    2    1       5   4   3   2
1 1 1 0            8    6    4    2       6   5   4   3
1 1 0 0           12    9    6    3       7   6   5   4
1 0 0 0           16   12    8    4       8   7   6   5
```

Reduction

```
          +/V               +/[1]M                +/[2]M
 10                   12   15   18         6   15   24
          ×/V               ×/[1]M                +/M
 24                   28   80   162        6   15   24
```

Inner Product

```
      M+.×M               M+.≤M              M+.×1 4 7
  30    36    42        3   3   3        30   66   102
  66    81    96        1   2   2                M+.×M[;1]
 102   126   150        0   0   1        30   66   102
(Ordinary Matrix
 Product)
```

```
(M+.×N)[I;J] is equivalent to +/M[I;]×M[;J]
```

Character arrays are specified by the use of quotation marks and behave like numeric arrays except that they are not in the domain of addition and other arithmetic functions:

```
      A←'DIGIT'
      A[1 2 3]
DIG
      A='I'
0  1  0  1  0
```

Monadic form fB		f	Dyadic form AfB	
Definition or example	**Name**		**Name**	**Definition or example**
+B ←→ 0+B	Plus	+	Plus	2+3.2 ←→ 5.2
-B ←→ 0-B	Negative	-	Minus	2-3.2 ←→ ¯1.2
×B ←→ (B>0)-(B<0)	Signum	×	Times	2×3.2 ←→ 6.4
÷B ←→ 1÷B	Reciprocal	÷	Divide	2÷3.2 ←→ 0.625
B ⌈B ⌊B 3.14 4 3 ¯3.14 ¯3 ¯4	Ceiling	⌈	Maximum	3⌈7 ←→ 7
	Floor	⌊	Minimum	3⌊7 ←→ 3
*B ←→ (2.71828..)*B	Exponential	*	Power	2*3 ←→ 8
⍟*N ←→ N ←→ *⍟N	Natural logarithm	⍟	Logarithm	A⍟B ←→ Log B base A A⍟B ←→ (⍟B)÷⍟A
∣¯3.14 ←→ 3.14	Magnitude	∣	Residue	Case — A∣B A≠0 — B-(∣A)×⌊B÷∣A A=0,B≥0 — B A=0,B<0 — Domain error
!0 ←→ 1 !B ←→ B×!B-1 or !B ←→ Gamma(B+1)	Factorial	!	Binomial coefficient	A!B ←→ (!B)÷(!A)×!B-A 2!5 ←→ 10 3!5 ←→ 10
?B ←→ Random choice from ⍳B	Roll	?	Deal	A Mixed Function (See Table 2)
○B ←→ B×3.14159...	Pi times	○	Circular	See Table at left
∼1 ←→ 0 ∼0 ←→1	Not	∼		
		∧	And	
		∨	Or	
		⍲	Nand	
		⍱	Nor	
		<	Less	Relations
		≤	Not greater	Result is 1 if the relation holds, 0 if it does not:
		=	Equal	3≤7 ←→ 1
		≥	Not less	7≤3 ←→ 0
		>	Greater	
		≠	Not Equal	

A	B	A∧B	A∨B	A⍲B	A⍱B
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

(-A)○B	A	A○B
(1-B*2)*.5	0	(1-B*2)*.5
Arcsin B	1	Sine B
Arccos B	2	Cosine B
Arctan B	3	Tangent B
(¯1+B*2)*.5	4	(1+B*2)*.5
Arcsinh B	5	Sinh B
Arccosh B	6	Cosh B
Arctanh B	7	Tanh B

Table of Dyadic ○ Functions

TABLE 1. PRIMITIVE SCALAR FUNCTIONS

Name	Sign[1]	Definition or example[2]
Size	ρA	ρP ↔ 4 ρE ↔ 3 4 ρ5 ↔ ι0
Reshape	VρA	Reshape A to dimension V 3 4ρι12 ↔ E 12ρE ↔ ι12 0ρE ↔ ι0
Ravel	,A	,A ↔ (×/ρA)ρA ,E ↔ι12 ρ,5 ↔ 1
Catenate	V,V	P,ι2 ↔ 2 3 5 7 1 2 'T','HIS' ↔ 'THIS'
Index[3,4]	V[A] M[A;A] A[A;.. ..;A]	P[2] ↔3 P[4 3 2 1] ↔7 5 3 2 E[1 3;3 2 1] ↔ 3 2 1 11 10 9 E[1;] ↔ 1 2 3 4 ABCD E[;1] ↔ 1 5 9 'ABCDEFGHIJKL'[E] ↔ EFGH IJKL
Index generator[3]	ιS	First S integers ι4 ↔ 1 2 3 4 ι0 ↔ an empty vector
Index of[3]	VιA	Least index of A in V, or 1+ρV Pι3 ↔2 PιE ↔ 5 1 2 5 3 5 4 5 5 5 5 5 4 4ι4 ↔ 1
Take	V↑A	Take or drop |V[I] first (V[I]≥0) or last (V[I]<0) elements of coordinate I 2 3↑X ↔ ABC EFG
Drop	V↓A	¯2↑P ↔ 5 7
Grade up[3,5]	⍋A	The permutation which would order A (ascending or descending) ⍋3 5 3 2 ↔ 4 1 3 2
Grade down[3,5]	⍒A	⍒3 5 3 2 ↔ 2 1 3 4
Compress[5]	V/A	1 0 1 0/P ↔ 2 5 1 0 1 0/E ↔ 1 3 5 7 9 11 1 0 1/[1]E ↔ 1 2 3 4 ↔ 1 0 1⌿E 9 10 11 12
Expand[5]	V\A	1 0 1\ι2 ↔ 1 0 2 1 0 1 1 1\X ↔ A BCD E FGH I JKL
Reverse[5]	⌽A	⌽X ↔ DCBA HGFE LKJI ⌽P ↔ 7 5 3 2 ⌽[1]X ↔ ⊖X ↔ IJKL EFGH ABCD
Rotate[5]	A⌽A	3⌽P ↔ 7 2 3 5 ↔ ¯1⌽P 1 0 ¯1⌽X ↔ BCDA EFGH LIJK
Transpose	V⍉A	Coordinate I of A becomes coordinate V[I] of result 2 1⍉X ↔ AEI BFJ CGK DHL 1 1⍉E ↔ 1 6 11
	⍉A	Transpose last two coordinates ⍉E ↔ 2 1⍉E
Membership	A∊A	ρW∊Y ↔ ρW P∊ι4 ↔ 1 1 0 0 E∊P ↔ 0 1 1 0 1 0 1 0 0 0 0 0
Decode	V⊥V	10⊥1 7 7 6 ↔ 1776 24 60 60⊥1 2 3 ↔ 3723
Encode	V⊤S	24 60 60⊤3723 ↔ 1 2 3 60 60⊤3723 ↔ 2 3
Deal[3]	S?S	W?Y ↔ Random deal of W elements from ιY

TABLE 2. PRIMITIVE MIXED FUNCTIONS (see notes on next page)

1. Restrictions on argument ranks are indicated by: *S* for scalar, *V* for vector, *M* for matrix, *A* for Any. Except as the first argument of *S*⍳*A* or *S*[*A*], a scalar may be used instead of a vector. A one-element array may replace any scalar.

2. Arrays used in examples:

				1	2	3	4			*ABCD*
P ↔	2 3 5 7		*E* ↔	5	6	7	8		*X* ↔	*EFGH*
				9	10	11	12			*IJKL*

3. Function depends on index origin.

4. Elision of any index selects all along that coordinate.

5. The function is applied along the last coordinate; the symbols ⌿, ⍀, and ⊖ are equivalent to /, \, and ⌽, respectively, except that the function is applied along the first coordinate. If [*S*] appears after any of the symbols, the relevant coordinate is determined by the scalar *S*.

Notes to Table 2

ELEMENTARY ALGEBRA

THE CONVENIENT USE OF ARRAYS IN APL MAKES IT EASY TO DISPLAY AND MANIPULATE MATHEMATICALLY MEANINGFUL PATTERNS. FOR EXAMPLE:

```
      2*2 3 4 5
4  8  16  32
      2*2 3 4 5 6 7
4  8  16  32  64  128
```

This pattern can be extended to the right by noting that each element is obtained by multiplying its predecessor by 2.

```
      2*¯2 ¯1 0 1 2 3
0.25  0.5  1  2  4  8
```

The pattern can be extended to the left by noting that each element is obtained by dividing its successor by 2. This gives a graphic picture of how meaning is assigned to zero and negative powers.

```
      4*1 2 3 4 5
4  16  64  256  1024
      4*1 1.5 2 2.5 3
4  8  16  32  64
      2*1 1.5 2 2.5 3
2  2.83  4  5.66  8
```

The same notions can be used to introduce fractional arguments.

FUNCTION TABLES (E.G., ADDITION TABLES, MULTIPLICATION TABLES, AND SUBTRACTION TABLES) CAN BE USED TO GIVE GRAPHIC PICTURES OF THE BEHAVIOR OF COMMON FUNCTIONS OF TWO ARGUMENTS:

```
      S←1 2 3 4 5 6 7
      S∘.+S
  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
  7   8   9  10  11  12  13
  8   9  10  11  12  13  14
```

```
      S∘.×S
  1   2   3   4   5   6   7
  2   4   6   8  10  12  14
  3   6   9  12  15  18  21
  4   8  12  16  20  24  28
  5  10  15  20  25  30  35
  6  12  18  24  30  36  42
  7  14  21  28  35  42  49
```

```
      S∘.⌈S
 1  2  3  4  5  6  7
 2  2  3  4  5  6  7
 3  3  3  4  5  6  7
 4  4  4  4  5  6  7
 5  5  5  5  5  6  7
 6  6  6  6  6  6  7
 7  7  7  7  7  7  7
```

```
      S∘.≥S
1 0 0 0 0 0 0
1 1 0 0 0 0 0
1 1 1 0 0 0 0
1 1 1 1 0 0 0
1 1 1 1 1 0 0
1 1 1 1 1 1 0
1 1 1 1 1 1 1
```

```
      S∘.-S
 0 ¯1 ¯2 ¯3 ¯4 ¯5 ¯6
 1  0 ¯1 ¯2 ¯3 ¯4 ¯5
 2  1  0 ¯1 ¯2 ¯3 ¯4
 3  2  1  0 ¯1 ¯2 ¯3
 4  3  2  1  0 ¯1 ¯2
 5  4  3  2  1  0 ¯1
 6  5  4  3  2  1  0
```

```
      S∘.<S
0 1 1 1 1 1 1
0 0 1 1 1 1 1
0 0 0 1 1 1 1
0 0 0 0 1 1 1
0 0 0 0 0 1 1
0 0 0 0 0 0 1
0 0 0 0 0 0 0
```

CERTAIN PROPERTIES OF FUNCTIONS (SUCH AS COMMUTATIVITY) CAN BE RELATED TO THE PATTERNS OBSERVED IN THEIR FUNCTION TABLES:

```
      S∘.-S
0 ¯1 ¯2 ¯3 ¯4 ¯5 ¯6
1  0 ¯1 ¯2 ¯3 ¯4 ¯5
2  1  0 ¯1 ¯2 ¯3 ¯4
3  2  1  0 ¯1 ¯2 ¯3
4  3  2  1  0 ¯1 ¯2
5  4  3  2  1  0 ¯1
6  5  4  3  2  1  0
```

```
      ⍉S∘.-S
 0  1  2  3  4  5  6
¯1  0  1  2  3  4  5
¯2 ¯1  0  1  2  3  4
¯3 ¯2 ¯1  0  1  2  3
¯4 ¯3 ¯2 ¯1  0  1  2
¯5 ¯4 ¯3 ¯2 ¯1  0  1
¯6 ¯5 ¯4 ¯3 ¯2 ¯1  0
```

The transpose of a function table is the table of the same function with the arguments commuted. Since the two tables do not agree, the subtraction function is not commutative.

```
      S∘.×S
1  2  3  4  5  6  7
2  4  6  8 10 12 14
3  6  9 12 15 18 21
4  8 12 16 20 24 28
5 10 15 20 25 30 35
6 12 18 24 30 36 42
7 14 21 28 35 42 49
```

```
      ⍉S∘.×S
1  2  3  4  5  6  7
2  4  6  8 10 12 14
3  6  9 12 15 18 21
4  8 12 16 20 24 28
5 10 15 20 25 30 35
6 12 18 24 30 36 42
7 14 21 28 35 42 49
```

The transpose of a table for a commutative function agrees with the original function.

```
      T←S-4
      T
¯3 ¯2 ¯1 0 1 2 3
      T∘.×T
 9  6  3 0 ¯3 ¯6 ¯9
 6  4  2 0 ¯2 ¯4 ¯6
 3  2  1 0 ¯1 ¯2 ¯3
 0  0  0 0  0  0  0
¯3 ¯2 ¯1 0  1  2  3
¯6 ¯4 ¯2 0  2  4  6
¯9 ¯6 ¯3 0  3  6  9
```

The function table for multiplication applied to both negative and positive arguments can be used to give some insight into the rules for the sign of a product.

THE FUNCTION TABLE FOR EQUALS (=) APPLIED TO THE VALUES OF A FUNCTION AND AN APPROPRIATE SET OF VALUES FROM THE RANGE OF THE FUNCTION YIELDS AN UNUSUAL INSIGHT INTO THE MEANING OF GRAPHS AND BAR CHARTS:

```
      ∇Z←F X
[1]   Z←(X-3)×(X-5)∇
```

The function F is a parabola with zeros at 3 and 5.

```
      S←1 2 3 4 5 6 7
```

```
      F S
8 3 0 ¯1 0 3 8
```

F S yields the values of the parabola for the argument S.

```
      R←8 7 6 5 4 3 2 1 0 ¯1
```

R is the range of values occurring in F S.

```
      R∘.=F S                 R∘.≤F S
1 0 0 0 0 0 1            1 0 0 0 0 0 1
0 0 0 0 0 0 0            1 0 0 0 0 0 1
0 0 0 0 0 0 0            1 0 0 0 0 0 1
0 0 0 0 0 0 0            1 0 0 0 0 0 1
0 0 0 0 0 0 0            1 0 0 0 0 0 1
0 1 0 0 0 1 0            1 1 0 0 0 1 1
0 0 0 0 0 0 0            1 1 0 0 0 1 1
0 0 0 0 0 0 0            1 1 0 0 0 1 1
0 0 1 0 1 0 0            1 1 1 0 1 1 1
0 0 0 1 0 0 0            1 1 1 1 1 1 1
```

The 1s represent a graph and a bar chart of F.

```
' *'[1+R∘.=F S]          ' *'[1+R∘.≤F S]

*     *                  *     *
                         *     *
                         *     *
                         *     *
                         *     *
 *   *                   **   **
                         **   **
                         **   **
  * *                    *** ***
   *                     *******
```

The asterisks represent a graph and a bar chart of F.

THE USE OF VECTORS PERMITS A CLEAR AND SIMPLE TREATMENT OF POLYNOMIALS:

```
      C←3 1 4 2                   Vector of coefficients.

      X←5                         Assigned argument value.

      E←0 1 2 3                   Vector of exponents.

      X*E                         Vector of powers of X.
1  5  25  125

      C×X*E                       Terms of the polynomial.
3  5  100  250

      +/C×X*E                     Sum of terms.
358

      ¯1+⍳⍴C                      Exponents appropriate to the
0  1  2  3                        coefficient vector C.

      +/C×X*¯1+⍳⍴C                General expression for any
358                               coefficient vector C.

      C←1 4 6 4 1
      +/C×X*¯1+⍳⍴C
1296

      ∇Z←C POL X                  Definition of a polynomial
[1]   Z←+/C×X*¯1+⍳⍴C∇             function.

      3 1 4 2 POL 5
358

      1 4 6 4 1 POL 2
81
```

THE COMPUTATION OF THE PRODUCT OF TWO POLYNOMIALS (I.E., THE COEFFICIENTS OF A POLYNOMIAL WHICH IS EQUIVALENT TO THE PRODUCT OF THE POLYNOMIALS) CAN BE STATED CLEARLY IN TERMS OF THE VECTORS OF COEFFICIENTS:

```
      C←3 1 4 2
      D←2 0 5 1 3

      C∘.×D
  6  0 15  3  9
  2  0  5  1  3
  8  0 20  4 12
  4  0 10  2  6
6  2  23  12  30  17  14  6

      E←6 2 23 12 30 17 14 6

      E POL 3
30432
      (C POL 3)×(D POL 3)
30432
```

This multiplication table contains the products of all pairs of coefficients.

A simple argument shows that they should be summed diagonally as indicated by the lines.

ALL STEPS OF A PROCESS CAN BE SHOWN CLEARLY IN APL. FOR EXAMPLE, THE SUMMATION OF THE COEFFICIENTS IN THE POLYNOMIAL PRODUCT (SHOWN INFORMALLY ON THE PRECEDING PAGE) CAN BE COMPLETED AS FOLLOWS:

```
      C←3 1 4 2
      D←2 0 5 1 3
      D,0×1↓C
2 0 5 1 3 0 0 0
      C∘.×D,0×1↓C
   6  0 15  3  9  0  0  0
   2  0  5  1  3  0  0  0
   8  0 20  4 12  0  0  0
   4  0 10  2  6  0  0  0
```

Append zeros to *D* so as to append zero columns to the multiplication table.

```
      1-⍳⍴C
0 ¯1 ¯2 ¯3
      (1-⍳⍴C)⌽C∘.×D,0×1↓C
   6  0 15  3  9  0  0  0
   0  2  0  5  1  3  0  0
   0  0  8  0 20  4 12  0
   0  0  0  4  0 10  2  6
```

Skew the table (by rotating the rows) so as to align in columns the coefficients to be added.

```
      +/[1](1-⍳⍴C)⌽C∘.×D,0×1↓C
6 2 23 12 30 17 14 6
```

Sum the columns to obtain the final result.

```
      ∇Z←C PROD D
[1]   Z←+/[1](1-⍳⍴C)⌽C∘.×D,0×1↓C∇
```

Define a polynomial product function.

```
      C PROD D
6 2 23 12 30 17 14 6
```

TABLES CAN ALSO BE USED TO ILLUMINATE NOTIONS NOT DIRECTLY RELATED TO THE TABLE OF A FUNCTION. FOR EXAMPLE, THE PRIME NUMBERS OR THE "PRIMENESS" OF A NUMBER CAN BE TREATED IN SEVERAL INTERESTING WAYS:

```
      S
1  2  3  4  5  6  7
0  1  1  0  1  0  1
```

The primeness of each element of *S* is indicated by the number 1 (for prime) or 0 (for not prime) appearing below it.

An expression for the primeness vector can be developed as follows:

```
      S∘.|S
0  0  0  0  0  0  0
1  0  1  0  1  0  1
1  2  0  1  2  0  1
1  2  3  0  1  2  3
1  2  3  4  0  1  2
1  2  3  4  5  0  1
1  2  3  4  5  6  0
```

Make a remainder table for a set of consecutive integers beginning with 1.

```
     0=S∘.|S
1 1 1 1 1 1 1
0 1 0 1 0 1 0
0 0 1 0 0 1 0
0 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
```

Compare the remainder table with 0 to obtain a "divisibility" table.

```
      +/[1]0=S∘.|S
1  2  2  3  2  4  2
```

Sum the columns of the divisibility table to obtain the number of divisors of each element of *S*.

```
      2=+/[1]0=S∘.|S
0  1  1  0  1  0  1
```

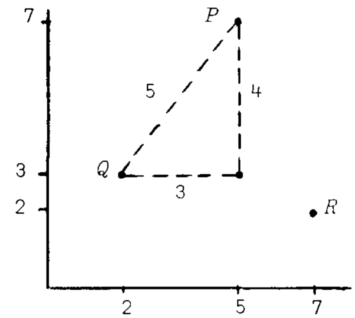
Compare the sums with 2 to determine primeness (since a prime has exactly two distinct divisors).

```
      U←2=+/[1]0=S∘.|S
      U                           The logical vector which
0  1  1  0  1  0  1               determines primeness
      U/S
2  3  5  7                        can be used to select the primes.


      ∇Z←PR N;S                   Define a function
[1]   Z←(2=+/[1]0=S∘.|S)/S←⍳N∇    to determine the primes
                                  up to N.
      PR 25
2  3  5  7  11  13  17  19  23


      S←2 3 4 5 6 7 8
      S∘.×S
   4   6   8  10  12  14  16      Alternatively, form a
   6   9  12  15  18  21  24      multiplication table (not
   8  12  16  20  24  28  32      including 1) and determine
  10  15  20  25  30  35  40      primeness by finding if the
  12  18  24  30  36  42  48      number does not occur in
  14  21  28  35  42  49  56      the table.
  16  24  32  40  48  56  64

      S∊S∘.×S
0  0  1  0  1  0  1
      ~S∊S∘.×S
1  1  0  1  0  1  0
      (~S∊S∘.×S)/S
2  3  5  7
```

COORDINATE GEOMETRY AND STATICS

```
      P←5 7
      Q←2 3
      R←7 2
```

Each point in a plane can be represented by a two-element vector of its coordinates.




```
      D←P-Q
      D
3 4
```

Displacement between P and Q.

```
      (+/D*2)*.5
5
```

Distance between P and Q.

```
      M←3 2ρP,Q,R
      M
  5  7
  2  3
  7  2
```

A set of N points (representing a triangle or other polygon) can be represented by an N by 2 matrix.

```
      1ϕ[1]M
  2  3
  7  2
  5  7
```

The same points "circulated."

```
      D←M-1ϕ[1]M
      D
  3  4
 ¯5  1
  2 ¯5
```

The displacements between each pair of points.

```
      L←(+/D*2)*.5
      L
5  5.099  5.385
```

The distances between each pair (i.e., the lengths of the sides of the triangle).

```
      S←.5×+/L
      S
7.742
```

The semi-perimeter of the triangle.

```
      (×/S,S-L)*.5
11.5
```

The area of the triangle by Hero's formula.

IN A SPACE OF THREE DIMENSIONS THE EXPRESSIONS FOR DISPLACEMENT, DISTANCE, ETC., ARE IDENTICAL WITH THOSE FOR 2-SPACE:

```
      P←5 7 2
      Q←2 3 14
      R←4.25 6 5

      D←P-Q
      D                           Displacement.
3  4  12

      (+/D*2)*.5                  Distance.
13

      M←3 3⍴P,Q,R                 A triangle in 3-space.
      M
     5          7          2
     2          3         14
     4.25       6          5

      1⌽[1]M
     2          3         14
     4.25       6          5
     5          7          2

      D←M-1⌽[1]M                  All displacements.
      D
   3         4        ¯12
  ¯2.25     ¯3          9
  ¯0.75     ¯1          3

      L←(+/D*2)*.5                All lengths.
      L
13  9.75  3.25

      S←.5×+/L                    Semi-perimeter.
      S
13

      (×/S,S-L)*.5                An area of zero implies that
0                                 the three points are collinear.
```

THE NOTIONS OF THE CENTER OF A FIGURE AND THE CENTER OF GRAVITY OF A SET OF POINT MASSES ARE EASILY EXPRESSED IN TERMS OF THE MATRIX OF COORDINATES:

```
      M←3 2⍴5 7 2 3 7 2       A triangle in 2-space.
      M
  5  7
  2  3
  7  2

      +/[1]M                  The "sum" of the points.
14  12
      (+/[1]M)÷3              The average (i.e., center) of the
4.667  4                      points.


      W←2 3 5                 The weights of masses at the
                              three points.

      W+.×M                   The total "moment" of the points.
51  33

      (W+.×M)÷+/W             The moment per unit weight, i.e.,
5.1  3.3                      the location of a single mass of
                              the same total weight to produce
                              the same moment.  This is the
                              center of gravity.

      (W÷+/W)+.×M             An equivalent statement of center
5.1  3.3                      of gravity, based on an obvious
                              mathematical identity.

      W÷+/W                   W÷+/W is a normalized mass, i.e.,
0.2  0.3  0.5                 it has a total mass of 1.
      +/(W÷+/W)
1

                              The same expressions apply to
                              3-space and to any number of
                              points.
```

DETERMINANTS IN THE COMPUTATION OF AREAS:

```
      M
  5  7
  2  3
  7  2
```

A triangle in 2-space

```
      1,M
  1  5  7
  1  2  3
  1  7  2
```

bordered by a column of 1s

```
      DET 1,M
23
```

yields a matrix whose determinant is twice the (signed) area of the triangle. (See Felix Klein, Elementary mathematics from an advanced standpoint: Geometry.)

```
      M[1 3 2;]
 5  7
 7  2
 2  3
      DET 1,M[1 3 2;]
¯23
```

The sign of the area is positive if the vertices occur in counter-clockwise order, and negative otherwise.

```
      N
   5       7
   2       3
   4.25    6
      DET 1,N
0
```

If the area is zero, the points are collinear. If the area is not zero, the sign tells whether the points are in clockwise order, and hence tells whether one point lies above or below the line joining the other two.

The definition of the determinant function itself can be briefly stated: the function *SDET* shows the essential scheme and *DET* contains some extra steps to take care of the occurrence of a zero in the upper left corner of the matrix.

```
      ∇Z←SDET M
[1]   Z←M[1;1]
[2]   →0×⍳∨/1=⍴M
[3]   Z←Z×SDET 1 1↓M-M[;1]∘.×M[1;]÷M[1;1]∇

      ∇Z←DET M;K
[1]   M[K,1;]←M[1,K←K⍳⌈/K←|M[;1];]
[2]   Z←(1E¯9<|M[1;1])×M[1;1]×¯1*K≠1
[3]   →0×⍳∨/(1=⍴M),0=Z
[4]   Z←Z×DET 1 1↓M-M[;1]∘.×M[1;]÷M[1;1]∇
```

THE SAME EXPRESSIONS APPLY TO THE VOLUME OF A TETRAHEDRON IN 3-SPACE, AND HENCE TO QUESTIONS OF THE POSITION OF A POINT RELATIVE TO THE PLANE DETERMINED BY THE THREE OTHER POINTS.

A tetrahedron in 3-space.

```
      M
4  8  3
2  4  9
6  4  5
6  9  4

      1,M
1  4  8  3
1  2  4  9
1  6  4  5
1  6  9  4

      DET 1,M
64
      DET 1,M[2 1 3 4;]
¯64

      N
1 0 0
0 1 0
0 0 1
0 0 0

      DET 1,N
¯1
```

Six times the signed volume of the tetrahedron. If the points are plotted in a right-handed coordinate system, then the sign is positive if the order of the first three points is counter-clockwise when viewed from the fourth point.

SOME USEFUL FUNCTIONS AND THE COMPUTATION OF *PI*:

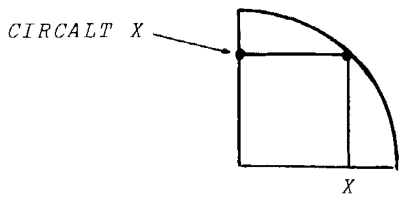
```
      ∇Z←D M
[1]   Z←¯1↓(+/(M-1⌽[1]M)*2)*.5∇
```

The distance between adjacent points of *M*.

```
      M
  1  4
  1  4
  7  6
  9  8
  5  1

      D M
0  6.325  2.828  8.062
```

Example to show how the function *D* works.

A function to compute the altitude of a point on the unit circle whose first coordinate is *X*:




```
      ∇Z←CIRCALT X
[1]   Z←(1-X*2)*.5∇

      CIRCALT .5 .6 .7 1
0.866  0.8  0.7141  0
```

```
      ∇Z←GRID N
[1]   Z←0,(⍳N)÷N∇

      GRID 5
0  0.2  0.4  0.6  0.8  1
```

A function to generate a set of points from 0 to 1 separated by an interval of 1÷*N*.

A function to approximate *PI* by twice the length of the sides of a portion of a polygon inscribed in the first quadrant of a circle:

```
      ∇Z←PI N
[1]   Z←2×+/D (GRID N),[1.5] CIRCALT GRID N∇

      PI 5
3.115105951
      PI 1000
3.141583356

      (GRID 5),[1.5] CIRCALT GRID 5
    0          1
    0.2        0.9798
    0.4        0.9165
    0.6        0.8
    0.8        0.6
    1          0
```

FINITE DIFFERENCES AND THE CALCULUS

```
      ∇Z←DIF V
[1]   Z←(1↓V)-¯1↓V∇

      X←0,⍳6
      V←X*2
      V
0  1  4  9  16  25  36
      1↓V
1  4  9  16  25  36
      ¯1↓V
0  1  4  9  16  25
```

```
      DIF V
1  3  5  7  9  11
```

First difference of the square function.

```
      DIF DIF V
2  2  2  2  2
```

Second difference of the square function.

```
      V←X*3
      V
0  1  8  27  64  125  216
```

```
      DIF V
1  7  19  37  61  91
```

First difference of the cubic function.

```
      DIF DIF V
6  12  18  24  30
```

Second difference of the cubic function.

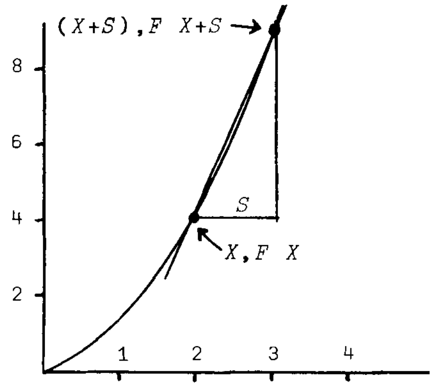
```
      DIF DIF DIF V
6  6  6  6
```

Third difference of the cubic function.

```
      DIF DIF DIF DIF X*4
24  24  24
```

Fourth difference of the quartic function.

THE SLOPE FUNCTION GIVING THE SLOPE OF THE SECANT THROUGH POINTS *X,F X* AND *(X+S),F X+S* YIELDS AN APPROXIMATION TO THE SLOPE OF THE TANGENT TO *F* AT THE POINT *X,F X* FOR *S* SMALL:




```
      ∇Z←F X
[1]   Z←X*2∇

      ∇Z←S SLOPE X
[1]   Z←((F X+S)-F X)÷S∇

      1 SLOPE 2
5
      .5 SLOPE 2
4.5
      .1 SLOPE 2
4.1

      .1 .01 .001 .0001 SLOPE 2
4.1  4.01  4.001  4.0001
```

The slope appears to approach a limit for small values of the spacing *S*.

```
      S←.000001
      (S,-S) SLOPE 2
4.000001  3.999999001
```

The same limit is approached for a negative value of *S*.

```
      S SLOPE X←1 2 3 4
2.000001  4.000001  6.000001  8.000001
      2×X
2  4  6  8
```

The slope at a set of points *X* is also a function (i.e., 2×*X*).

EXPERIMENTATION WITH THE SLOPE FUNCTION APPLIED TO VARIOUS FUNCTIONS CAN LEAD TO CONJECTURES CONCERNING THE TANGENT SLOPE (I.E., DERIVATIVE) FOR VARIOUS FUNCTIONS:

```
      ∇Z←F X
[1]   Z←X*3∇

      S SLOPE X
3.000003  12.000006  27.000009  48.000012
      3×X*2
3  12  27  48

      ∇Z←F X
[1]   Z←X*4∇

      S SLOPE X
4.000006  32.000024  108.00005  256.0001
      4×X*3
4  32  108  256

      ∇Z←F X                          ∇Z←C POLY X
[1]   Z←C POLY X∇              [1]    X←(X∘.*¯1+⍳⍴C)+.×C∇

      C←3 1 2 4

      S SLOPE X
17.000014  57.000026  121.00004  209.00005
```

```
      (1↓C×¯1+⍳⍴C) POLY X
17  57  121  209
```

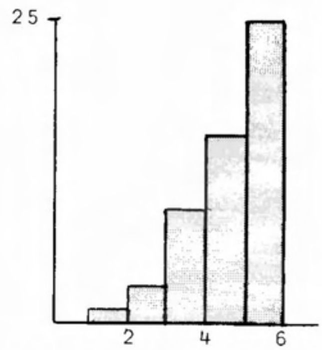
A polynomial equivalent to the derivative of the original polynomial.

```
      ¯1+⍳⍴C
0  1  2  3
      C×¯1+⍳⍴C
0  1  4  12
      1↓C×¯1+⍳⍴C
1  4  12
```

Determination of the coefficients of the derived polynomial.

(APPROXIMATE) INTEGRATION BY THE RECTANGULAR RULE CAN BE REPRESENTED AS A LINEAR FUNCTION (MATRIX PRODUCT) WHOSE INVERSE IS SEEN TO BE A DIFFERENCING OF THE RESULT:

```
      X←1 2 3 4 5
      V←X*2
      V
1  4  9  16  25
      A
 1 0 0 0 0
 1 1 0 0 0
 1 1 1 0 0
 1 1 1 1 0
 1 1 1 1 1
```




```
      R←A+.×V
      R
1  5  14  30  55
```

Multiplication by the "accumulator" matrix *A* yields the approximate integrals over 1, 2, 3, 4, and 5 intervals.

```
      (⌹A)+.×R
1  4  9  16  25
```

Multiplication of the result by the inverse matrix ⌹*A* yields the original values.

```
      D←⌹A
      D
 1  0  0  0  0
¯1  1  0  0  0
 0 ¯1  1  0  0
 0  0 ¯1  1  0
 0  0  0 ¯1  1
```

The matrix ⌹*A* is seen to be a "difference" matrix whose application is equivalent to differencing (except that it includes in the result the first element of the argument).

```
      ∇Z←DIF X
[1]   Z←(1↓X)-¯1↓X∇

      DIF R
4  9  16  25
      1↓D+.×R
4  9  16  25
      D+.×V
1  3  5  7  9
      A+.×D+.×V
1  4  9  16  25
      A+.×V
1  5  14  30  55
      D+.×A+.×V
1  4  9  16  25
```

Differencing and integration are inverse and may be applied in either order.

IF DIFFERENCING IS REPRESENTED AS A LINEAR FUNCTION THEN THE EFFECT OF REPEATED DIFFERENCING CAN EASILY BE SHOWN IN TERMS OF THE ORIGINAL ARGUMENT AND APPEARS AS ALTERNATING BINOMIAL COEFFICIENTS:

```
      V
1  4  9  16  25
      D
 1  0  0  0  0
¯1  1  0  0  0
 0 ¯1  1  0  0
 0  0 ¯1  1  0
 0  0  0 ¯1  1
```

```
      D+.×V
1  3  5  7  9
```

First difference.

```
      D+.×(D+.×V)
1  2  2  2  2
```

Second difference.

```
      (D+.×D)+.×V
1  2  2  2  2
```

An equivalent statement for second differences.

```
      D+.×D
 1  0  0  0  0
¯2  1  0  0  0
 1 ¯2  1  0  0
 0  1 ¯2  1  0
 0  0  1 ¯2  1
```

The matrix which yields second differences.

```
      D+.×D+.×D
 1  0  0  0  0
¯3  1  0  0  0
 3 ¯3  1  0  0
¯1  3 ¯3  1  0
 0 ¯1  3 ¯3  1
```

The matrix for third differences.

THE SLOPE FUNCTION CAN BE TREATED SIMILARLY (AS A LINEAR FUNCTION) TO EXHIBIT THE INVERSE RELATIONSHIP BETWEEN DIFFERENTIATION AND INTEGRATION:

```
      S←.1
      X←3+S×1 2 3 4 5
      X
3.1  3.2  3.3  3.4  3.5
      V←X*2
      V
9.61  10.24  10.89  11.56  12.25
```

```
      D
 1  0  0  0  0
¯1  1  0  0  0
 0 ¯1  1  0  0
 0  0 ¯1  1  0
 0  0  0 ¯1  1
```

```
      D+.×V
9.61  0.63  0.65  0.67  0.69
```

First difference of *V*.

```
      (D+.×V)÷S
96.1  6.3  6.5  6.7  6.9
```

The slopes between points of *V*.

```
      (D÷S)+.×V
96.1  6.3  6.5  6.7  6.9
```

An equivalent expression for slope.

```
      R←(D÷S)+.×V
```

Points of the slope function.

```
      (⌹D÷S)+.×R
9.61  10.24  10.89  11.56  12.25
      V
9.61  10.24  10.89  11.56  12.25
```

The inverse function applied to the slope function *R* yields the original values *V*.

```
     ⌹D      ||        ⌹D÷S                    ||       S×⌹D
1 0 0 0 0    ||  0.1  0.0  0.0  0.0  0.0       ||  0.1  0.0  0.0  0.0  0.0
1 1 0 0 0    ||  0.1  0.1  0.0  0.0  0.0       ||  0.1  0.1  0.0  0.0  0.0
1 1 1 0 0    ||  0.1  0.1  0.1  0.0  0.0       ||  0.1  0.1  0.1  0.0  0.0
1 1 1 1 0    ||  0.1  0.1  0.1  0.1  0.0       ||  0.1  0.1  0.1  0.1  0.0
1 1 1 1 1    ||  0.1  0.1  0.1  0.1  0.1       ||  0.1  0.1  0.1  0.1  0.1
```

```
      S×(⌹D)+.×R
9.61  10.24  10.89  11.56  12.25
```

This form of the inverse is clearly equivalent to integration by the rectangular rule (i.e., the *K*th row of *D* sums over the first *K* points, and the multiplication by *S* accounts for the grid size.

LOGIC

LOGIC CONCERNS PROPOSITIONS. A PROPOSITION IS ANY STATEMENT WHICH MAY BE JUDGED TRUE OR FALSE, I.E., A PROPOSITION IS A FUNCTION WITH A RANGE OF TWO ELEMENTS. THESE ELEMENTS MAY BE REPRESENTED IN A VARIETY OF WAYS, USUALLY BY THE WORDS TRUE AND FALSE OR BY THE NUMBERS 1 AND 0:

```
      X←3               Propositions read as:

      X<5               X is less than 5
1                       true

      X>5               X is greater than 5
0                       false

      0=3|X             X is divisible by 3
1                       true

      (X>5)∧0=3|X       X is greater than 5 and X is divisible by 3
0                       false
```

```
      X←1 2 3 4 5 6 7 8 9 10
      X<5
1 1 1 1 0 0 0 0 0 0
      X>5
0 0 0 0 0 1 1 1 1 1
      0=3|X
0 0 1 0 0 1 0 0 1 0
      (X>5)∧0=3|X
0 0 0 0 0 1 0 0 1 0
      (X<5)/X
1 2 3 4
      (X>5)/X
6 7 8 9 10
      (0=3|X)/X
3 6 9
      ((X>5)∧0=3|X)/X
6 9
```

A proposition applied to a vector yields a logical vector. This logical vector is, in effect, the characteristic vector (with respect to the universe X) of the set of elements which satisfy the proposition, i.e., for which the proposition is true.

The result of the proposition applied to X can therefore be used to select that subset of X defined by the proposition.

THE PROPOSITION (X>5)∧0=3|X IS SAID TO BE *COMPOUND* BECAUSE IT IS FORMED AS A FUNCTION (∧) OF SIMPLER PROPOSITIONS (X>5) AND (0=3|X). A FUNCTION SUCH AS ∧ (PRONOUNCED *AND*) WHICH IS DEFINED ONLY ON THE ARGUMENTS 0 AND 1 IS CALLED A *LOGICAL* OR *BOOLEAN* FUNCTION. THE COMPLETE BEHAVIOR OF A LOGICAL FUNCTION CAN BE EXHIBITED AS A 2-BY-2 FUNCTION TABLE AS FOLLOWS:

```
      L←0 1
      L∘.∧L
0 0
0 1
```

```
∧|0 1
0|0 0
1|0 1
```

THERE IS ONE FURTHER FAMILIAR LOGICAL FUNCTION ∨ (*OR*) AND TWO LESS FAMILIAR FUNCTIONS ⍲ (*NOT-AND*) AND ⍱ (*NOT-OR*):

```
      L∘.∨L
0 1
1 1
```

```
      L∘.⍲L
1 1
1 0
```

```
      L∘.⍱L
1 0
0 0
```

WHEN APPLIED ONLY TO LOGICAL ARGUMENTS (0 OR 1), THE RELATIONS (< ≤ = ≥ > ≠) ARE IN EFFECT LOGICAL FUNCTIONS (SINCE THEIR RANGE IS ALSO 0 1) AND ARE OFTEN GIVEN SPECIAL NAMES WHEN USED IN THIS WAY. FOR EXAMPLE:

Exclusive-Or

```
      L∘.≠L
0 1
1 0
```

Material Implication

```
      L∘.≤L
1 1
0 1
```

Identity

```
      L∘.=L
1 0
0 1
```

```
      X
1  2  3  4  5  6  7  8  9  10  11  12
      ((0=2|X)∧0=3|X)≤0=6|X
1  1  1  1  1  1  1  1  1  1  1  1
```

X is divisible by 2 and X is divisible by 3 *implies* that X is divisible by 6.

A THEOREM IS A PROPOSITION WHICH IS CLAIMED TO BE UNIVERSALLY TRUE, I.E., TO HAVE THE VALUE 1 WHEN APPLIED TO ANY ELEMENT IN THE UNIVERSE OF DISCOURSE. FOR EXAMPLE, THE PROPOSITION

$$((0=2|X)\wedge(0=3|X))\leq 0=6|X$$

IS A THEOREM WHICH MAY BE VERBALIZED IN A VARIETY OF WAYS:

> X is divisible by 2 and X is divisible by 3 implies that X is divisible by 6.
>
> Any number divisible by both 2 and 3 is also divisible by 6.
>
> If X is divisible by both 2 and 3 then X is divisible by 6.
>
> Divisibility by 2 and 3 implies divisibility by 6.

PROPOSITIONS ARE ALSO USED IN THE DEFINITION OF SETS, AND EXAMPLES MAY BE FOUND IN THE ACCOMPANYING DISCUSSION OF SETS.

SINCE A LOGICAL FUNCTION APPLIES TO TWO ARGUMENTS EACH CHOSEN FROM THE DOMAIN 0 1, THE SET OF ALL POSSIBLE ARGUMENTS CAN BE LISTED AS THE ROWS OF A 4 BY 2 MATRIX AS FOLLOWS:

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

THIS MATRIX (AND ANALOGOUS MATRICES OF DIMENSION 2⋆N BY N) CAN BE PRODUCED BY THE FOLLOWING "TRUTH TABLE" FUNCTION:

```
      ∇Z←T N
[1]   Z←⍉⌊(Nρ2)⊤¯1+⍳2⋆N∇

      T 2                    T 3
 0  0                    0  0  0
 0  1                    0  0  1
 1  0                    0  1  0
 1  1                    0  1  1
                         1  0  0
                         1  0  1
                         1  1  0
                         1  1  1

      ⍉T 2
  0  0  1  1
  0  1  0  1

      ⍉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

      ⍉T 4
  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
```

EACH OF THE LOGICAL FUNCTIONS (∧, ∨, ≠, ETC.) CAN BE APPLIED TO ROWS OF THE TABLE *T* 2 TO YIELD THE VECTOR OF ALL POSSIBLE VALUES OF THE FUNCTION:

```
      T 2
  0  0
  0  1
  1  0
  1  1

      ∧/T 2              ∨/T 2              ≠/T 2
 0  0  0  1        0  1  1  1        0  1  1  0
```

EACH OF THESE VECTORS IS CALLED THE CHARACTERISTIC VECTOR OF THE CORRESPONDING FUNCTION. TABLES OF THESE FUNCTIONS CAN THEREFORE BE PRODUCED BY APPENDING THEIR CHARACTERISTIC VECTORS AS COLUMNS TO THE MATRIX *T* 2:

```
      (((T 2),∧/T 2),∨/T 2),≠/T 2
 0  0 | 0  0  0
 0  1 | 0  1  1
 1  0 | 0  1  1
 1  1 | 1  1  0
```

SINCE ANY FOUR-ELEMENT LOGICAL VECTOR IS A CHARACTERISTIC VECTOR OF SOME LOGICAL FUNCTION, THERE ARE IN ALL 2*4 LOGICAL FUNCTIONS, AND THEY ALL OCCUR AS COLUMNS IN THE FOLLOWING MATRIX:

```
      ⍉T 4
  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
  0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
  0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
```

THE CHARACTERISTIC VECTORS OF THE FUNCTIONS ∧, ∨, AND ≠ CAN BE SEEN TO OCCUR AS COLUMNS 2, 8, AND 7 OF THE FOREGOING TABLE.

THE FUNCTION TABLE FOR ALL POSSIBLE LOGICAL FUNCTIONS OF TWO ARGUMENTS CAN THEREFORE BE EXHIBITED AS FOLLOWS:

```
      (T 2),⍉T 4
0  0 | 0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
0  1 | 0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1
1  0 | 0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1
1  1 | 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
```

THE TABLE OF ARGUMENTS FOR (I.E., THE DOMAIN OF) ALL LOGICAL FUNCTIONS OF THREE ARGUMENTS IS GIVEN BY THE FOLLOWING MATRIX:

```
      T 3
0  0  0
0  0  1
0  1  0
0  1  1
1  0  0
1  0  1
1  1  0
1  1  1
```

THE TABLE OF ALL CHARACTERISTIC VECTORS FOR 3 ARGUMENTS WOULD THEREFORE BE GIVEN BY ⍉T 8 AND WOULD CONTAIN 2*8 COLUMNS. A PORTION OF THE FUNCTION TABLE FOR 3 ARGUMENTS (REPRESENTING THE FIRST 17 FUNCTIONS) CAN THEREFORE BE DISPLAYED AS FOLLOWS:

```
      (T 3),8 17↑⍉T 8
0  0  0 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  1 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  1  0 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  1  1 | 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
1  0  0 | 0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  0
1  0  1 | 0  0  0  0  1  1  1  1  0  0  0  0  1  1  1  1  0
1  1  0 | 0  0  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0
1  1  1 | 0  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1  0
```

SETS

```
      A←2 3 5 7 11
      B←6 2 8 4
```

A finite set can be represented by a list of its elements.

```
      3∊A
1
      3∊B
0
```

Membership is the fundamental function defined on a set.

```
      (3∊A)∧3∊B
0
```

Does 3 belong to *A* and to *B*.

```
      (3∊A)∨3∊B
1
```

Does 3 belong to *A* or to *B*.

```
      (3∊A)∧~3∊B
1
```

Does 3 belong to *A* and not to *B*.

```
      U←⍳12
      U
1 2 3 4 5 6 7 8 9 10 11 12
```

The universe of discourse is the set of all possible elements under consideration.

```
      A∊U
1 1 1 1 1
      B∊U
1 1 1 1
```

Every element of any set in the universe belongs to *U*.

```
      U∊A
0 1 1 0 1 0 1 0 0 0 1 0
      U∊B
0 1 0 1 0 1 0 1 0 0 0 0
```

The logical vector that shows which elements of *U* belong to *A* is called the characteristic vector of *A* (with respect to the universe *U*).

```
      (U∊A)/U
2 3 5 7 11
```

Compression of *U* by the characteristic vector of *A* yields *A*.

```
      (U∊A)∧U∊B
0 1 0 0 0 0 0 0 0 0 0 0
      ((U∊A)∧U∊B)/U
2
```

The characteristic vector of the set of elements which belong to both *A* and *B*.

ANY PROPOSITION (I.E., ANY FUNCTION WHOSE RANGE IS THE SET 0 1)
DEFINES A SET:

```
      ∇Z←P X
[1]    Z←(X≥3)∧(X<11)∇

      U
1 2 3 4 5 6 7 8 9 10 11 12
      P U
0 0 1 1 1 1 1 1 1 1 0 0
      SP←(P U)/U
      SP
3 4 5 6 7 8 9 10
```

The proposition *P* applied to the universe *U* yields the characteristic vector of the set of all elements of *U* which satisfy the proposition. The expression (*P U*)/*U* therefore yields the set of all such elements.

```
      ∇Z←Q X
[1]    Z←0=2|X∇
      Q U
0 1 0 1 0 1 0 1 0 1 0 1
      SQ←(Q U)/U
      SQ
2 4 6 8 10 12
```

Proposition defining the set of all even integers.

```
      (P U)∧Q U
0 0 0 1 0 1 0 1 0 1 0 0
      ((P U)∧Q U)/U
4 6 8 10
```

The characteristic vector and the set of all elements which belong to both *SP* and *SQ*, i.e., the <u>intersection</u> of *SP* and *SQ*.

```
      ((P U)∨Q U)/U
2 3 4 5 6 7 8 9 10 12
```

The <u>union</u> of *SP* and *SQ*.

```
      (P U)∧~Q U
0 0 1 0 1 0 1 0 1 0 0 0
      ((P U)∧~Q U)/U
3 5 7 9
```

The characteristic vector and the set of all elements which belong to *SP* and not to *SQ*.

IF *P* IS A PROPOSITION AND *SP* IS THE SET IT DEFINES WITH RESPECT TO THE UNIVERSE *U*, THEN THE MEMBERSHIP OF ANY ELEMENT *X* CAN BE DETERMINED EITHER BY THE EXPRESSION *P X* OR BY THE EXPRESSION *X∊SP*:

```
      ∇Z←P X
[1]   Z←(X≥3)∧(X<11)∇

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

      SP←(P U)/U
      SP
3  4  5  6  7  8  9  10

      X←5
      P X
1

      X∊SP
1

      P 2
0

      2∊SP
0

      X←1 2 3 4 5

      (P X)=X∊SP
1  1  1  1  1
```

AN INFINITE SET (SUCH AS THE SET OF ALL POSITIVE EVEN INTEGERS) CANNOT BE REPRESENTED BY A LIST OF ITS ELEMENTS, BUT CAN STILL BE REPRESENTED BY A PROPOSITION. IT IS NOT POSSIBLE TO APPLY THE PROPOSITION TO THE ENTIRE INFINITE UNIVERSE, BUT MEMBERSHIP OF ANY ELEMENT OR FINITE COLLECTION OF ELEMENTS CAN BE DETERMINED BY APPLYING THE PROPOSITION TO THEM:

```
      ∇Z←PEI X
[1]   Z←(X>0)∧0=2|X∇
```

A proposition which defines the set of positive even integers.

```
      PEI 4
1

      PEI ¯4
0

      PEI 2.4
0

      X←0 1 2 3 4 5

      PEI X
0 0 1 0 1 0

      (PEI X)/X
2 4
```

FUNCTIONS FOR INTERSECTION, DIFFERENCE, UNION AND SET EQUALITY CAN EASILY BE DEFINED:

```
      ∇Z←A I B
[1]   Z←(A∊B)/A∇

      ∇Z←A D B
[1]   Z←(~A∊B)/A∇

      ∇Z←A U B
[1]   Z←A,B D A∇

      ∇Z←A EQ B
[1]   Z←∧/(A∊B),B∊A∇

      A←1 2 3 4 5
      B←2 4 6 8

      A I B
2 4
      B I A
2 4
      A EQ B
0
      (A I B) EQ (B I A)
1
      A D B
1 3 5
      A U B
1 2 3 4 5 6 8
```

These functions apply equally to sets of characters:

```
      E←'ABCDE'
      F←'BDFH'

      E I F
BD
      E U F
ABCDEFH
      E D F
ACE
```

ALL 2⋆N SUBSETS OF A SET OF N ELEMENTS CAN BE NEATLY REPRESENTED BY THE MATRIX OF THEIR CHARACTERISTIC VECTORS. THIS MATRIX CAN ALSO BE CONCEIVED AS THE N-DIGIT BINARY REPRESENTATIONS OF THE INTEGERS FROM 0 TO ¯1+2⋆N, AND CAN THEREFORE BE PRODUCED BY THE FOLLOWING FUNCTION:

```
      ∇Z←T N
[1]   Z←(Nρ2)⊤¯1+⍳2⋆N∇

      T 2
  0  0  1  1
  0  1  0  1
      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

      S←'ABC'
      Z←T ρS
      Z
  0  0  0  0  1  1  1  1
  0  0  1  1  0  0  1  1
  0  1  0  1  0  1  0  1

      Z[;4]
0  1  1
      Z[;4]/S
BC
```

The characteristic vector of the fourth subset, and the set itself.

```
      R←2 3 5
      R+.×Z
0  5  3  8  2  7  5  10
```

The sums over all subsets of the set R.

```
      R×.⋆Z
1  5  3  15  2  10  6  30
```

The products over all subsets of the set R. (These are the symmetric products occurring in Newton's identities for the coefficients of a polynomial in terms of its roots R.)

PROPOSITIONS DEFINING VARIOUS SETS OF NUMBERS (SUCH AS PRIMES AND PERFECT SQUARES) CAN BE CONVENIENTLY STATED AND USED:

A proposition for the primes.

```
      ∇Z←PP S
[1]   Z←2=+/[1]0=(⍳⌈/S)∘.|S∇

      S←5+⍳9
      S
6  7  8  9  10  11  12  13  14
      PP S
0  1  0  0  0  1  0  1  0
      (PP S)/S
7  11  13
```

A proposition for squares.

```
      ∇Z←PSQ S
[1]   Z←(S*.5)=⌊S*.5∇
      PSQ S
0  0  0  1  0  0  0  0  0
      (PSQ S)/S
9
```

A proposition to determine whether a given vector represents possible lengths for the sides of a polygon.

```
      ∇Z←PPOL L
[1]   Z←∧/L<.5×+/L∇

      PPOL 5 2 4
1
      PPOL 5 2 2
0
      PPOL 3 1 7 4
1
      PPOL 3 1 8 4
0
```

ELECTRIC CIRCUITS

ARRAYS ARE USEFUL IN THE TREATMENT OF ELECTRICAL CIRCUITS FOR SEVERAL REASONS:

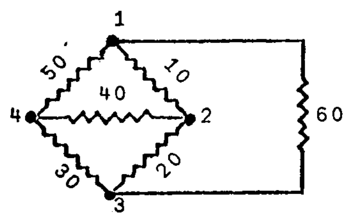
1. A circuit is composed of a set of elements whose characteristics can therefore be described by a vector or other array, e.g.:

   ```
   R←10 20 30 40 50 60
   ```

   might describe a set of six resistors having resistances of 10, 20, 30, 40, 50, and 60 ohms.

2. The topology of the circuit (i.e., the connections of the circuit elements (branches) with the nodes) can be described by various arrays. For example, the topology of the accompanying circuit (formed from *R*) can be described by the following branch connection matrix:

   ```
           BC
   1  2  3  4  4  1
   2  3  4  2  1  3
   ```

   whose *I*th column shows the nodes from and to which the *I*th element is connected. (A direction is specified even though it is immaterial for bilateral elements such as resistors.)

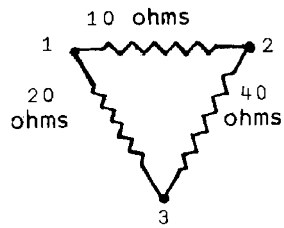
3. Most circuits are (approximately) linear (that is, voltages are linear functions of currents, and vice versa) and relations among them are easily represented as matrix products.

SIMPLE SERIES AND PARALLEL CIRCUITS

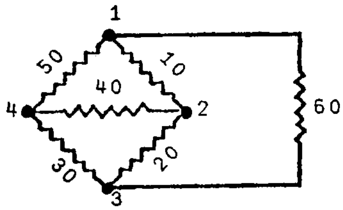
```
      R←10 20 30 40              Values for 4 resistors (in ohms).

      +/R                        Resistance of a series circuit.
100
      ÷R                         Conductances.
0.1  0.05  0.0333  0.025
      +/÷R                       Conductance of a parallel circuit.
0.208
      ÷+/÷R                      Resistance of a parallel circuit.
4.8
      L←1 2 3 4                  Four inductances.

      A←100                      Angular velocity (2 pi times
                                 frequency).

      A×L                        Inductive reactance.
100  200  300  400

      -÷A×L                            Inductive susceptance.
¯0.01  ¯0.005  ¯0.00333  ¯0.0025

      C←5 6 7 8                  Four capacitors.


      A×C                        Capacitive susceptance.
5000  6000  7000  8000

      M←3 4⍴(÷R,L),C                   Description of four elements
      M                                each comprising resistance,
0.1000  0.0500  0.0333  0.0250         capacitance, and inductance.
1.0000  0.5000  0.3333  0.2500
5.0000  6.0000  7.0000  8.0000

      Q←2 3⍴1 0 0 0,(-÷A),A            Determination of a complex
      Q                                admittance matrix for the
  1.00   0.00    0.00                  four elements at velocity A
  0.00  ¯0.01  100.00                  with the parts of each
                                       element in parallel.

      Q+.×M
    0.100     0.050     0.033     0.025
  499.990   599.995   699.997   799.997
```

BECAUSE THE RELATION BETWEEN VOLTAGES AND CURRENTS IS LINEAR, THE NODE CURRENTS *I* IN A CIRCUIT CAN BE DETERMINED FROM THE NODE VOLTAGES *V* BY (INNER PRODUCT) MULTIPLICATION BY A SUITABLE ADMITTANCE MATRIX *Y* AND THE VOLTAGES CAN BE OBTAINED AS *V*←*Z*+.×*I*, WHERE *Z* IS A SUITABLE IMPEDANCE MATRIX. FOR EXAMPLE:

```
      Y
 0.150 ¯0.100 ¯0.050
¯0.100  0.125 ¯0.025
¯0.050 ¯0.025  0.075

      Z
 14.286  11.429   0.000
 11.429  17.143   0.000
  0.000   0.000   0.000

      V←4 5 0
      I←Y+.×V
      I
0.1  0.225  ¯0.325
      Z+.×I
4  5  0
```

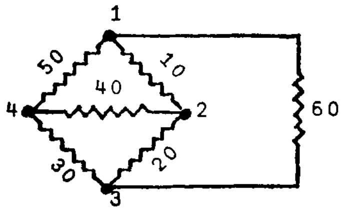


SIMILARLY:




```
      Y
 0.1367 ¯0.1000 ¯0.0167 ¯0.0200
¯0.1000  0.1750 ¯0.0500 ¯0.0250
¯0.0167 ¯0.0500  0.1000 ¯0.0333
¯0.0200 ¯0.0250 ¯0.0333  0.0783

      Z
 17.9700  12.9784   9.4841   0.0000
 12.9784  16.0399  10.1830   0.0000
  9.4841  10.1830  16.6722   0.0000
  0.0000   0.0000   0.0000   0.0000

      V←1 2 3 0
      I←Y+.×V
      I
¯0.113  0.1  0.183  ¯0.17
      Z+.×I
1  2  3  0
```

THE ADMITTANCE MATRIX *Y* CAN EASILY BE DETERMINED FROM THE COMPONENT ADMITTANCE MATRIX *CAM* (WHOSE DIAGONAL CONTAINS THE ADMITTANCE OF THE COMPONENTS) AND THE INCIDENCE MATRIX *E* WHOSE *J*TH ROW SHOWS CONNECTIONS FROM (DENOTED BY 1) AND TO (DENOTED BY ¯1) EACH OF THE BRANCHES (I.E., COMPONENTS) ASSOCIATED WITH THE VARIOUS COLUMNS. FOR EXAMPLE:




```
      CAM
0.100  0.000  0.000  0.000  0.000  0.000
0.000  0.050  0.000  0.000  0.000  0.000
0.000  0.000  0.033  0.000  0.000  0.000
0.000  0.000  0.000  0.025  0.000  0.000
0.000  0.000  0.000  0.000  0.020  0.000
0.000  0.000  0.000  0.000  0.000  0.017
```

```
      Y←E+.×CAM+.×⍉E
      Y
 0.137 ¯0.100 ¯0.017 ¯0.020
¯0.100  0.175 ¯0.050 ¯0.025
¯0.017 ¯0.050  0.100 ¯0.033
¯0.020 ¯0.025 ¯0.033  0.078
```

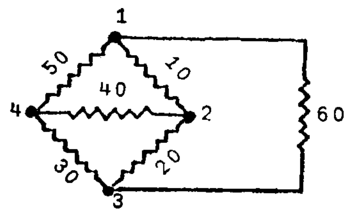
```
      E
 1  0  0  0 ¯1  1
¯1  1  0 ¯1  0  0
 0 ¯1  1  0  0 ¯1
 0  0 ¯1  1  1  0
```

Since the admittance matrix is singular, the impedance matrix is obtained as the (bordered) inverse of a submatrix of *Y*:

```
      Z←(ρY)↑⌹(¯1+ρY)↑Y
      Z
17.970  12.978   9.484   0.000
12.978  16.040  10.183   0.000
 9.484  10.183  16.672   0.000
 0.000   0.000   0.000   0.000
```

FUNCTIONS RELATING THE TWO IMPORTANT REPRESENTATIONS OF THE TOPOLOGY OF A CIRCUIT (THE BRANCH CONNECTION MATRIX *BC* AND THE INCIDENCE MATRIX *E*) ARE EASILY DEFINED:

```
      ∇E←F BC
[1]   E←-/(⍳⌈/,BC)∘.=⍉BC∇

      ∇BC←G E
[1]   BC←(1 ¯1∘.=⍉E)+.×⍳1↑⍴E∇

      BC
  1  2  3  4  4  1
  2  3  4  2  1  3

      E←F BC
      E
  1  0  0  0 ¯1  1
 ¯1  1  0 ¯1  0  0
  0 ¯1  1  0  0 ¯1
  0  0 ¯1  1  1  0

      G E
  1  2  3  4  4  1
  2  3  4  2  1  3
```



THE BRANCH CURRENTS AND VOLTAGES *BI* AND *BV* ARE EASILY SEEN TO BE RELATED TO THE NODE CURRENTS AND VOLTAGES *I* AND *V* BY THE INCIDENCE MATRIX *E* AS FOLLOWS:

```
      CAM
  0.100  0.000  0.000  0.000  0.000  0.000
  0.000  0.050  0.000  0.000  0.000  0.000
  0.000  0.000  0.033  0.000  0.000  0.000
  0.000  0.000  0.000  0.025  0.000  0.000
  0.000  0.000  0.000  0.000  0.020  0.000
  0.000  0.000  0.000  0.000  0.000  0.017

      E                                   V
  1  0  0  0 ¯1  1              1  2  3  0
 ¯1  1  0 ¯1  0  0
  0 ¯1  1  0  0 ¯1
  0  0 ¯1  1  1  0

      BV←V+.×E               Kirchhoff's Voltage Law
      BV
¯1  ¯1  3  ¯2  ¯1  ¯2
      BI←CAM+.×BV            Kirchhoff's Current Law
      I←E+.×BI
      I
¯0.1133  0.1  0.1833  ¯0.17
```

The branch voltages can also be obtained from the equivalent expression

```
      BV←(⍉E)+.×V
```

Collecting these results yields:

```
      I←E+.×CAM+.×(⍉E)+.×V
```

From this it is clear why

```
      Y←E+.×CAM+.×⍉E
```

yields an admittance matrix *Y* such that

```
      I←Y+.×V
```

All preceding results apply to a component admittance matrix with non-zero off-diagonal elements and hence can treat circuits with <u>active</u> elements represented as "voltage-controlled current sources."

THE COMPUTER: A DEVICE FOR THE AUTOMATIC EXECUTION OF ALGORITHMS

It is best to approach the study of the internal structure of any device with previous knowledge of the <u>function</u> of the device, that is, of how to use it and of <u>what</u> it does as opposed to <u>how</u> it does it. The function of a computer is to execute algorithms presented to it in a manner familiar to anyone who knows how to write and enter programs.

For example, if the following characters are entered:

```
      X←1
      Z←(X+2)×(X+4)
      Z
```

the computer will act to assign the value 1 to the name *X*, the value 15 to *Z*, and to print the number 15. The computer can therefore be conceived as a function which produces these results when applied to the argument *P*, where *P* is the following matrix of characters:

```
      P←3 13⍴'X←1          Z←(X+2)×(X+4)Z            '
      P
X←1
Z←(X+2)×(X+4)
Z
```

The computer can therefore be represented by the following function:

```
      ∇COMP P
[1]   IC←1                Instruction counter set to 1.
[2]   IR←P[IC;]           Instruction fetched into instruction
                           register.
[3]   ⍎IR                 Instruction in IR executed.
[4]   IC←IC+1             Instruction counter incremented.
[5]   →(IC∊⍳1↑⍴P)/2∇      Repeat for next instruction if any
                           remain.

      COMP P              Use of the computer.
15
      X
1
```

```
      ∇COMP P
[1]   IC←1
[2]   IR←P[IC;]
[3]   ⍎IR
[4]   IC←IC+1
[5]   →(IC∊⍳1↑⍴P)/2∇
```

The function *COMP* displays the sequence (instruction fetch, instruction execution, updating of instruction counter) which is fundamental to any computer. It displays this clearly by subordinating (through the use of the execution function ⍎) the details of the execution of individual instructions. These details can then be brought out in a sequence of simple steps so as to make clear the complete structure of the computer.

However, the simple function *COMP* does not handle all programs, and we will first illustrate how its capability can be extended by showing a modification necessary to handle branching:

```
      ∇ COMP2 P
[1]   IC←1
[2]   IR←P[IC;]
[3]   →(IR[1]='→')/8
[4]   ⍎IR
[5]   IC←IC+1
[6]   →(IC∊⍳1↑⍴P)/2
[7]   →0
[8]   IC←⍎1↓IR
[9]   →6∇
```

```
        P2
X←1
Z←(X+2)×(X+4)
Z
X←X+1
→2×X≤4
```

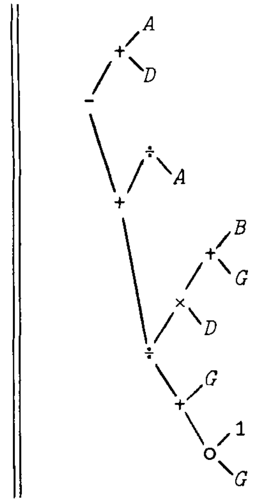
A program which employs branching.

Lines 8 and 9 are executed to respecify *IC* (that is, branch) if the first character of the instruction is →.

```
      COMP2 P
15
      COMP2 P2
15
24
35
48
```

AN IMPORTANT STEP IN EXPOSING THE DETAILS OF EXECUTION IS THE <u>COMPILATION</u> OF A COMPOUND EXPRESSION SUCH AS (*A*+*D*)-(÷*A*)+((*B*+*G*)×*D*)÷*G*+1○*G* INTO AN EQUIVALENT SEQUENCE OF SIMPLE EXPRESSIONS. THIS WILL BE SHOWN AS A SEQUENCE OF THREE TRANSFORMATIONS:

```
      S3←'(A+D)-(÷A)+((B+G)×D)÷G+1○G'
      D3←PARSE S3
      D3
    A
  +
    D
-

    ÷
      A
  +
          B
        +
          G
      ×
        D
    ÷
        G
      +
          1
        ○
          G
```



A diagramming or <u>parse</u> of the expression in which the result is a character matrix (in this case 19 by 11) which exhibits the sequence of execution in the form of a <u>tree</u>. The lines drawn in the copy of *D*3 on the right show this structure more clearly.

```
      P3←POLISH D3
      P3
-+AD+÷ A÷×+BGD+G○1G
```

The parenthesis-free or <u>Polish</u> form of the expression represents a dyadic function such as *A*×*B* by ×*AB*, and a monadic function such as ÷*A* analogously with a blank space for the non-existent left argument, that is, ÷ *A*.

```
      C3←COMPILE P3
      C3
Z←1○G
A←G+Z
B←B+G
C←B×D
D←C÷A
E← ÷A
F←E+D
G←A+D
G-F
```

This final sequence of simple statements employs names for each of the partial results. (The use of underscored names avoids conflict with the names in the original expression.)

```
      G←1+D←1+B←1+A←1
      A,B,D,G
1  2  3  4
      COMP 1 26⍴S3
¯2.550078271
      COMP C3
¯2.550078271
```

The assignment of values to the variables *A*, *B*, *D*, and *G* permits both the original expression *S*3 and the compiled form *C*3 to be executed by the computer *COMP*. (The expression *S*3 must be reformed to a 1-row matrix to be acceptable as an argument for *COMP*.)

THE TREATMENT OF AN EXPRESSION WHICH INCLUDES ASSIGNMENTS (←) IS SHOWN BELOW:

```
       S4
Z←X×Y←G+D

       PARSE S4
   Z
←
     X
  ×
       Y
    ←
         G
       +
         D

       POLISH PARSE S4
←Z×X←Y+GD

       COMPILE POLISH PARSE S4
U←G+D
Y←U
W←X×Y
Z←W

       X←1+G←1+D←1
       D,G,X
1  2  3

       Y
VALUE ERROR
       Y
       ^
       Z
VALUE ERROR
       Z
       ^
       COMP COMPILE POLISH PARSE S4
       Y
3
       Z
9
       )ERASE Y Z

       COMP 1 9ρS4
       Y
3
       Z
9
```

THE *PARSE* FUNCTION EMPLOYS THREE MAJOR FUNCTIONS *C*, *L*, AND *R* WHICH RESPECTIVELY SELECT THE CENTRAL FUNCTION (I.E., THE OVERALL FUNCTION WHICH IS TO BE EXECUTED LAST) OF THE EXPRESSION, THE PART TO THE LEFT OF THE CENTRAL FUNCTION, AND THE PART TO THE RIGHT:

```
    ∇ Z←C E
[1]   Z←E[CENTRALFN E]
    ∇
    ∇ Z←L E
[1]   Z←(¯1+CENTRALFN E)↑E
    ∇
    ∇ Z←R E
[1]   Z←(CENTRALFN E)↓E
    ∇
```

```
      S3
(A+D)-(÷A)+((B+G)×D)÷G+1○G
      C S3
-
      L S3
(A+D)
      R S3
(÷A)+((B+G)×D)÷G+1○G
      L R S3
(÷A)
```

THESE FUNCTIONS IN TURN EMPLOY THE FUNCTIONS *CENTRALFN* (WHICH DETERMINES THE INDEX OF THE CENTRAL FUNCTION), *DEPTH* (WHICH DETERMINES THE DEPTH IN PARENTHESES OF EACH PART OF AN EXPRESSION), AND *FUNCTIONS* (WHICH DETERMINES WHICH CHARACTERS IN AN EXPRESSION REPRESENT FUNCTIONS):

```
    ∇ Z←CENTRALFN E
[1]   Z←((FUNCTIONS E)∧0=DEPTH E)⍳1
    ∇
    ∇ Z←DEPTH E
[1]   Z←+\(E='(')-0,¯1↓E=')'
    ∇
    ∇ Z←FUNCTIONS E
[1]   Z←E∊'←+-×÷<≤=≥>≠∨∧?∊⍴~↑↓⍳○*⍟⌈⌊⊥⊤|'
    ∇
```

```
      E←L R R S3
      E
((B+G)×D)
      D←DEPTH E
      D
1 2 2 2 2 2 1 1 1
      '012'[1+D]
122222111
      FUNCTIONS E
0 0 0 1 0 0 1 0 0
```

THE *PARSE* FUNCTION EMPLOYS TWO FURTHER FUNCTIONS *STRIP* (WHICH STRIPS OFF OUTER PARENTHESES), AND *ON* (WHICH STACKS THE ROWS OF ONE TABLE ON TOP OF THE ROWS OF ANOTHER):

```
    ∇ Z←PARSE E
[1]   →0×⍳∧/~FUNCTIONS Z←STRIP E
[2]   Z←(' ',' ',PARSE L Z) ON(C Z) ON ' ',' ',PARSE R Z
    ∇
```

```
    ∇ Z←STRIP E
[1]   →0×⍳1≠⌊/DEPTH Z←E
[2]   Z←STRIP 1↓¯1↓E
    ∇
```

```
    ∇ Z←A ON B
[1]   A←(¯2↑1 1,⍴A)⍴A
[2]   B←(¯2↑1 1,⍴B)⍴B
[3]   Z←(((⍴A)⌈0 1×⍴B)↑A),[1]((⍴B)⌈0 1×⍴A)↑B
    ∇
```

```
      ⎕←F←STRIP E
(B+G)×D
      ⎕←A←PARSE L F
  B
+
  G
      ⎕←B←PARSE R F
D
```

```
      A ON B
  B
+
  G
D
```

```
      (' ',' ',A) ON C F
    B
  +
    G
×
```

THE *POLISH* FUNCTION FIRST STRIPS ALL BLANK COLUMNS FROM THE PARSED MATRIX *M*, AND THEN APPLIES THE FUNCTIONS *LT*, *CT*, AND *RT* TO SELECT THE LEFT, CENTER, AND RIGHT PARTS OF THE ARGUMENT, THE CENTER BEING DETERMINED AGAIN AS THE OVERALL FUNCTION:

```
    ∇ Z←POLISH M
[1]   Z←CT M←(∨/[1]M≠' ')/M
[2]   →0×⍳1≥1↑⍴M
[3]   Z←Z,(POLISH LT M),POLISH RT M
    ∇

    ∇ Z←CT M
[1]   Z←,1 1↑(' '≠FIRSTCOL M)/[1]M
    ∇

    ∇ Z←RT M
[1]   Z←(∨\¯1⌽' '≠FIRSTCOL M)/[1]M
    ∇

    ∇ Z←LT M
[1]   Z←(~∨\' '≠FIRSTCOL M)/[1]M
    ∇
```

```
      E
((B+G)×D)
      ⎕←M←PARSE E
    B
  +
    G
×
  D
      LT M
    B
  +
    G
      CT M
×
      RT M
  D

      ⎕←I←POLISH LT M
+BG
      ⎕←J←CT M
×
      ⎕←K←POLISH RT M
D
      J,I,K
×+BGD
      POLISH M
×+BGD
```

THE FUNCTION *FIRSTCOL* SELECTS THE FIRST COLUMN OF ITS ARGUMENT:

```
    ∇ Z←FIRSTCOL M
[1]   Z←,((1↑⍴M),1)↑M
    ∇
```

THE *COMPILE* FUNCTION ALSO EMPLOYS LEFT, RIGHT, AND CENTER FUNCTIONS (*LE*, *RE*, AND *CE*), THE CENTER BEING DETERMINED AS THE RIGHTMOST FUNCTION IN THE POLISH STRING AND THE TWO CHARACTERS FOLLOWING IT, I.E., THE SUBEXPRESSION WHICH IS TO BE EXECUTED FIRST:

```
    ∇ Z←CENTER E
[1]   Z←(LOCCENTER E)/E
    ∇
    ∇ Z←LEFT E
[1]   Z←(~∨\LOCCENTER E)/E
    ∇
    ∇ Z←RIGHT E
[1]   Z←LOCCENTER E
[2]   Z←(~Z∨∧\~Z)/E
    ∇
    ∇ Z←LOCCENTER E
[1]   Z←(ιρE)∊0 1 2+(FUNCTIONS E)⌈.×ιρE
    ∇
```

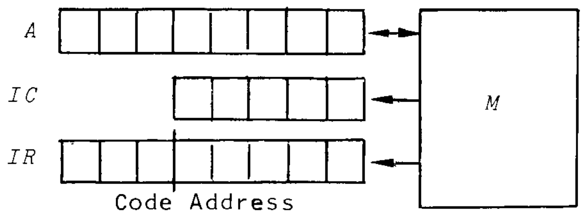
```
      F
(B+G)×D
      P←POLISH PARSE F
      P
×+BGD

      ⎕←LE←LEFT P
×
      ⎕←CE←CENTER P
+BG
      ⎕←RE←RIGHT P
D
      LOCCENTER P
0 1 1 1 0
```

THE *COMPILE* FUNCTION RE-ORDERS THE CENTER TO PRODUCE A NORMAL DYADIC EXPRESSION AND PREFIXES IT BY AN ITERMEDIATE NAME (CHOSEN FROM *NAMES*) AND AN ASSIGNMENT ARROW, BUT ONLY IF THE CENTER NEITHER CONTAINS AN ASSIGNMENT ARROW ITSELF NOR EXHAUSTS THE EXPRESSION:

```
      NAMES
ABCDEFGHIJKLMNOPQRSTUVWXYZ

    ∇ Z←COMPILE E;CE
[1]   CE←CENTER E
[2]   Z←((('←'∊CE)⍱3≥ρE)/NAMES[1],'←'),CE[2 1 3]
[3]   NAMES←1⌽NAMES
[4]   →0×ι3≥ρE
[5]   Z←Z ON COMPILE(LEFT E),Z[1],RIGHT E
    ∇
      CE
+BG
      CE[2 1 3]
B+G
      Z←NAMES[1],'←',CE[2 1 3]
      Z
A←B+G
      LE,Z[1],RE
×AD
      COMPILE LE,Z[1],RE
A×D
      Z ON COMPILE LE,Z[1],RE
A←B+G
A×D
```

A COMPUTER MAY ALSO BE TREATED AT A LEVEL OF DETAIL WHICH MAKES EXPLICIT THE BINARY REPRESENTATION OF NUMBERS AND INSTRUCTIONS. FOR EXAMPLE, A COMPUTER WITH THE FOLLOWING STRUCTURE AND INSTRUCTIONS CAN BE REPRESENTED BY THE FUNCTION *MACHINE* SHOWN BELOW:

Code	Instructions
000	Load *A*
001	Store *A*
010	Enter
011	Print
100	Add
101	Constant to *A*
110	Terminate
111	Branch if *A*≠*M*[31;]



A: Accumulator
IC: Instruction Counter
IR: Instruction Register
M: Memory of 32 8-bit words

```
      ∇MACHINE
[1]    IC←0 0 0 0 0
[2]    IR←M[2⊥IC;]
[3]    IC←(5ρ2)⊤1+2⊥IC
[4]    →5+2⊥3↑IR
[5]    →2,A←M[2⊥3↓IR;]
[6]    →2,M[2⊥3↓IR;]←A
[7]    →2,M[2⊥3↓IR;]←(8ρ2)⊤⎕
[8]    →2,⎕←2⊥M[2⊥3↓IR;]
[9]    →2,A←A PLUS M[2⊥3↓IR;]
[10]   →2,A←0 0 0,3↓IR
[11]   →0
[12]   (∧/A=M[31;])/2
[13]   →2,IC←3↓IR∇
```

L
S
E
P
A
C
T
B

Fetch instruction.
Increment *IC*.
Branch to execute instruction indicated by code in first three positions of *IR*.

```
      ∇Z←X PLUS Y
[1]    Z←(8ρ2)⊤(2⊥X)+2⊥Y∇
```

Addition function (detailed later).

(0-origin indexing is used in these functions, that is, the rows of *M* are indexed by the values 0, 1, 2, ...,31.)

IF THE FOLLOWING PROGRAM IS STORED IN THE COMPUTER (I.E., THE MEMORY IS INITIALLY SET TO THE INDICATED VALUE) THEN THE MACHINE (I.E., THE FUNCTION *MACHINE*) WILL COMPUTE AND PRINT THE SEQUENCE OF FIBONACCI NUMBERS, WHICH BEGINS WITH 1 1 AND CONTINUES WITH EACH NUMBER BEING THE SUM OF THE TWO PRECEDING IT. THE TABLE *P* AT THE RIGHT DISPLAYS THE MEANING OF EACH OF THE INSTRUCTION CODES IN THE MEMORY:

M		*P*
1 0 1 0 0 0 0 1	Constant 1 to *A*	*C* 1
0 0 1 1 1 1 1 1	Store *A* in 31	*S V*
0 0 1 1 1 1 1 0	Store *A* in 30	*S X*
0 0 1 1 1 1 0 1	Store *A* in 29	*S Y*
0 0 0 1 1 1 1 0	Load *A* from 30	*L X*
1 0 0 1 1 1 0 1	Add from 29	*A Y*
0 0 1 1 1 1 0 0	Store *A* in 28	*S Z*
0 1 1 1 1 1 1 0	Print from 30	*P X*
0 0 0 1 1 1 0 1	Load *A* from 29	*L Y*
0 0 1 1 1 1 1 0	Store *A* in 30	*S X*
0 0 0 1 1 1 0 0	Load *A* from 28	*L Z*
0 0 1 1 1 1 0 1	Store *A* in 29	*S Y*
1 1 1 0 0 1 0 0	Branch to 4	*B* 4
0 0 0 0 0 0 0 0	This row and succeeding rows are immaterial except that the last row should be all zero.	

```
     ∇ MACHINE
[1]    IC← 0 0 0 0 0
[2]    IR←M[2⊥IC;]
[3]    IC←(5⍴2)⊤1+2⊥IC
[4]    →5+2⊥3↑IR
[5]    →2,A←M[2⊥3↓IR;]
[6]    →2,M[2⊥3↓IR;]←A
[7]    →2,M[2⊥3↓IR;]←(8⍴2)⊤⎕
[8]    →2,⎕←2⊥M[2⊥3↓IR;]
[9]    →2,A←A PLUS M[2⊥3↓IR;]
[10]   →2,A← 0 0 0 ,3↓IR
[11]   →0
[12]   →2×⍳∧/A=M[31;]
[13]   →2,IC←3↓IR
     ∇
```

THE FOLLOWING TRACE OF THE EXECUTION OF THE FUNCTION *MACHINE* SHOWS THE DETAILED EXECUTION OF A PORTION OF THE PROGRAM STORED IN THE MEMORY *M*:

```
      )ORIGIN 0
WAS 1
      T∆MACHINE←ι13

      MACHINE
MACHINE[1] 0  0  0  0  0
MACHINE[2] 1  0  1  0  0  0  0  1
MACHINE[3] 0  0  0  0  1
MACHINE[4] 10
MACHINE[10] 2  0  0  0  0  0  0  0  1
MACHINE[2] 0  0  1  1  1  1  1  1
MACHINE[3] 0  0  0  1  0
MACHINE[4] 6
MACHINE[6] 2  0  0  0  0  0  0  0  1
MACHINE[2] 0  0  1  1  1  1  1  0
MACHINE[3] 0  0  0  1  1
MACHINE[4] 6
MACHINE[6] 2  0  0  0  0  0  0  0  1
MACHINE[2] 0  0  1  1  1  1  0  1
MACHINE[3] 0  0  1  0  0
MACHINE[4] 6
MACHINE[6] 2  0  0  0  0  0  0  0  1
MACHINE[2] 0  0  0  1  1  1  1  0
MACHINE[3] 0  0  1  0  1
MACHINE[4] 5
MACHINE[5] 2  0  0  0  0  0  0  0  1
MACHINE[2] 1  0  0  1  1  1  0  1
MACHINE[3] 0  0  1  1  0
MACHINE[4] 9
MACHINE[9] 2  0  0  0  0  0  0  1  0
MACHINE[2] 0  0  1  1  1  1  0  0
MACHINE[3] 0  0  1  1  1
MACHINE[4] 6
MACHINE[6] 2  0  0  0  0  0  0  1  0
MACHINE[2] 0  1  1  1  1  1  1  0
MACHINE[3] 0  1  0  0  0
MACHINE[4] 8
1
MACHINE[8] 2  1
MACHINE[2] 0  0  0  1  1  1  0  1
MACHINE[3] 0  1  0  0  1
MACHINE[4] 5
MACHINE[5] 2  0  0  0  0  0  0  0  1
```

THE MATRIX *P* SHOWN TO THE RIGHT OF THE PROGRAM FOR THE FIBONACCI NUMBERS IS AN EQUIVALENT SYMBOLIC STATEMENT WHICH IS EASIER TO WRITE. AN <u>ASSEMBLER</u> PROGRAM CALLED *ASSEMBLE* PRODUCES THE MATRIX *M* AS A FUNCTION OF *P*:

```
      )ORIGIN 1
WAS 0
      M←ASSEMBLE P
      M                                    P
 1 0 1 0 0 0 0 1                 C 1
 0 0 1 1 1 1 1 1                 S V
 0 0 1 1 1 1 1 0                 S X
 0 0 1 1 1 1 0 1                 S Y
 0 0 0 1 1 1 1 0                 L X
 1 0 0 1 1 1 0 1                 A Y
 0 0 1 1 1 1 0 0                 S Z
 0 1 1 1 1 1 1 0                 P X
 0 0 0 1 1 1 0 1                 L Y
 0 0 1 1 1 1 1 0                 S X
 0 0 0 1 1 1 0 0                 L Z
 0 0 1 1 1 1 0 1                 S Y
 1 1 1 0 0 1 0 0                 B 4
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0

      )ORIGIN 0
WAS 1
      MACHINE
1
1
2
3
5
8
```

THE ASSEMBLY PROGRAM IS SHOWN BELOW:

```
      ∇Z←ASSEMBLE P
[1]   Z←32 8ρ0
[2]   ST←0 6ρ' '                              Initialize symbol table.
[3]   I←0
[4]   →0×ι(1↑ρP)<I←I+1
[5]   INST←P[I;1]]
[6]   ARG←P[I;3]
[7]   Z[I;]←(CODE INST),BINARY NUMERIC ARG   Assemble Ith instruction.
[8]   →4×ιINST∈'BC'
[9]   ST←ST WITH ARG                          Add any new argument to symbol table.
[10]  Z[I;3+ι5]←ST ADDRESS ARG                Replace address part from symbol table if
[11]  →4∇                                      neither branch nor constant.

      ∇Z←CODE X
[1]   Z←2 2 2⊤¯1+'LSEPACTB'ιX∇               Encode symbols L, S, etc.

      ∇Z←BINARY X
[1]   Z←(5ρ2)⊤X∇                              X in 5 digit binary.

      ∇Z←NUMERIC X
[1]   Z←¯1+'0123456789'ιX∇                    Numeric equivalent of character vector.

      ∇Z←ST WITH NEW
[1]   Z←ST
[2]   →0×ι∨/NEW=ST[;1]                        Add NEW to symbol table if not already in it.
[3]   Z←ST,[1]NEW,CHAR BINARY 31-(ρST)[1]∇   Assign next address (in decreasing sequence).

      ∇Z←ST ADDRESS X
[1]   Z←NUMERIC 1↓ST[ST[;1]ιX;]∇              Address associated with name in symbol table.

      ∇Z←CHAR X
[1]   Z←'0123456789'[1+X]∇                    Character equivalent of numeric vector.
```

THE PROGRAM GIVEN FOR THE FIBONACCI NUMBERS WILL NEVER STOP. A MORE SATISFACTORY PROGRAM WHICH ACCEPTS AN ENTRY FROM THE KEYBOARD TO DETERMINE THE NUMBER OF FIBONACCI NUMBERS TO BE PRINTED IS SHOWN BELOW:

```
                              )ORIGIN 1
                        WAS 0
                              M←ASSEMBLE P2
      P2                      M
C 0                      1 0 1 0 0 0 0 0
S V                      0 0 1 1 1 1 1 1
E Q                      0 1 0 1 1 1 1 0
C 1                      1 0 1 0 0 0 0 1
S X                      0 0 1 1 1 1 0 1
S Y                      0 0 1 1 1 1 0 0
L X                      0 0 0 1 1 1 0 1
A Y                      1 0 0 1 1 1 0 0
S Z                      0 0 1 1 1 0 1 1
P X                      0 1 1 1 1 1 0 1
L Y                      0 0 0 1 1 1 0 0
S X                      0 0 1 1 1 1 0 1
L Z                      0 0 0 1 1 0 1 1
S Y                      0 0 1 1 1 1 0 0
C 1                      1 0 1 0 0 0 0 1
A V                      1 0 0 1 1 1 1 1
S V                      0 0 1 1 1 1 1 1
L Q                      0 0 0 1 1 1 1 0
B 6                      1 1 1 0 0 1 1 0
T                        1 1 0 1 1 0 1 0
                         0 0 0 0 0 0 0 0
                              . . .

                              )ORIGIN 0
                         WAS 1
                              MACHINE
                         ⎕:
                              4
                         1
                         1
                         2
                         3
                              MACHINE
                         ⎕:
                              6
                         1
                         1
                         2
                         3
                         5
                         8
```

The function *PLUS* used in conjunction with the function *MACHINE* adds two numbers which are represented in binary and yields their sum also represented in binary:

```
      ∇Z←X PLUS Y
[1]   Z←(8⍴2)⊤(2⊥X)+2⊥Y∇
```

This function does not show any of the detail necessary for designing a mechanical adder which would have to act on the individual digits of the representation. The design of such an adder can be approached by first treating a familiar representation (base 10), then the base 2 representation using addition of single digits, then the base 2 representation using only logical functions:

```
      ∇Z←X DPLUS Y               Decimal plus.
[1]   Z←X                        Sum (or addend) to result.
[2]   →(∧/0=Y)/0                 Stop if augend is zero.
[3]   X←10|Z+Y                   Sum without carry.
[4]   Y←1⌽10≤Z+Y                 New carry.
[5]   →1∇                        Repeat.

      T∆DPLUS←1 3 4
      1 9 9 DPLUS 0 0 1
DPLUS[1] 1  9  9                 A trace of the function DPLUS
DPLUS[3] 1  9  0                 shows its execution in detail.
DPLUS[4] 0  1  0
DPLUS[1] 1  9  0
DPLUS[3] 1  0  0
DPLUS[4] 1  0  0
DPLUS[1] 1  0  0
DPLUS[3] 2  0  0
DPLUS[4] 0  0  0
DPLUS[1] 2  0  0
2  0  0
```

THE FIRST FUNCTION FOR BINARY ADDITION (*BPLUS*) IS IDENTICAL TO THE FUNCTION FOR DECIMAL ADDITION EXCEPT THAT REMAINDERS AND CARRIES ARE TAKEN WITH RESPECT TO 2 RATHER THAN 10. THE SECOND FUNCTION (*LPLUS*) REPLACES THE RADIX 2 REMAINDERS AND CARRIES BY EQUIVALENT LOGICAL FUNCTIONS:

```
      ∇Z←X DPLUS Y           ∇Z←X BPLUS Y           ∇Z←X LPLUS Y
[1]   Z←X              [1]   Z←X              [1]   Z←X
[2]   →(∧/0=Y)/0       [2]   →(∧/0=Y)/0       [2]   →(∧/0=Y)/0
[3]   X←10|Z+Y         [3]   X←2|Z+Y          [3]   X←Z≠Y
[4]   Y←1⌽10≤Z+Y       [4]   Y←1⌽2≤Z+Y        [4]   Y←1⌽Z∧Y
[5]   →1∇              [5]   →1∇              [5]   →1∇
```

```
      X←(8⍴2)⊤199
      Y←(8⍴2)⊤1
      X
1 1 0 0 0 1 1 1
      Y
0 0 0 0 0 0 0 1
      1 9 9 DPLUS 0 0 1
2 0 0
      10⊥1 9 9 DPLUS 0 0 1
200
      X BPLUS Y
1 1 0 0 1 0 0 0
      2⊥X BPLUS Y
200
      2⊥X LPLUS Y
200
      T∆BPLUS←1 3 4
      X BPLUS Y
BPLUS[1] 1 1 0 0 0 1 1 1
BPLUS[3] 1 1 0 0 0 1 1 0
BPLUS[4] 0 0 0 0 0 0 1 0
BPLUS[1] 1 1 0 0 0 1 1 0
BPLUS[3] 1 1 0 0 0 1 0 0
BPLUS[4] 0 0 0 0 0 1 0 0
BPLUS[1] 1 1 0 0 0 1 0 0
BPLUS[3] 1 1 0 0 0 0 0 0
BPLUS[4] 0 0 0 0 1 0 0 0
BPLUS[1] 1 1 0 0 0 0 0 0
BPLUS[3] 1 1 0 0 1 0 0 0
BPLUS[4] 0 0 0 0 0 0 0 0
BPLUS[1] 1 1 0 0 1 0 0 0
1 1 0 0 1 0 0 0
```

BIBLIOGRAPHY

1. Berry, P. C., A. D. Falkoff and K. E. Iverson, Using the Computer to Compute: A Direct but Neglected Approach to Teaching Mathematics, IBM Philadelphia Scientific Center Technical Report No. 320-2988, May 1970.

2. Berry, P.C., G. Bartoli, C. Dell'Aquila and V. Spadavecchia, APL and Insight: Using Functions to Represent Concepts in Teaching, IBM Philadelphia Scientific Center Technical Report No. 320-3009, December 1971.

3. Falkoff, A. D. and K. E. Iverson, APL\360 User's Manual, IBM Corporation, 1968.

4. Iverson, K. E., Elementary Functions: an algorithmic treatment, Science Research Associates, Chicago, 1966.

5. Iverson, K. E., The Use of APL in Teaching, IBM Corporation, 1969.

6. Iverson, K. E., Elementary Algebra, IBM Philadelphia Scientific Center Technical Report No. 320-3001, June 1971.

7. Pakin, S., APL\360 Reference Manual, Science Research Asssociates, Chicago, 1968.

TECHNICAL REPORT INDEXING INFORMATION

1. AUTHOR(S): Iverson, K. E.	9. INDEX TERMS FOR THE IBM SUBJECT INDEX: APL Algebra Calculus Education Geometry Logic 05-Computer Application 16-Mathematics 21-Programming
2. TITLE: APL in Exposition	
3. ORIGINATING DEPARTMENT: Philadelphia Scientific Center	
4. REPORT NUMBER: 320-3010	
5a. NO. OF PAGES: 66	5b. NO. OF REFERENCES: 7

6a. DATE COMPLETED	6b. DATE OF INITIAL PRINTING	6c DATE OF LAST PRINTING
January 1972	January 1972	

7. ABSTRACT:

This report illustrates the use of APL for exposition in the teaching of various topics. The first section presents the characteristics of the language, and each of the succeeding sections illustrates its use in the presentation of material in some one discipline.

The treatment of each topic is self-contained, and so brief that it can only suggest the convenience provided by APL in more extended discussion. A perusal of several topics will illustrate the fact that the convenience of APL is not confined to any particular field. More extended use of the language is illustrated by some of the items in the bibliography.

This paper arose from material developed for a series of talks given at various locations over the past year or so. Its form betrays this origin; each page is relatively self-contained and is suitable for use as a transparency on an overhead projector.

8. DISTRIBUTION LIMITATIONS:

IBM

IBM Cambridge Scientific Center
545 Technology Square
Cambridge, Massachusetts 02139

IBM Houston Scientific Center
6900 Fannin Street
Houston, Texas 77025

IBM Los Angeles Scientific Center
1930 Century Park W.
Los Angeles, California 90067

IBM Palo Alto Scientific Center
2670 Hanover Street
Palo Alto, California 94304

IBM Philadelphia Scientific Center
3401 Market Street
Philadelphia, Pennsylvania 19104