APL\?

Roger K. W. Hui

Kenneth E. Iverson

E. E. McDonnell

Arthur T. Whitney

This paper describes a version of APL based upon the dictionary [1], but significantly simplified and enhanced, and directly usable on any machine that provides ASCII characters. It also describes salient features of a C implementation that has been tested on several machines, and is available as freeware. There have been four primary motivations for this work:

1. To provide an APL system for use in teaching mathematics and related topics that is modern, free, and transportable.

2. To devise a spelling scheme based on the ASCII alphabet that preserves the major advantages of the one-letter words based on the special alphabet commonly used in APL.

3. To exploit the advantages of breaking from the strict conformance with earlier APL that is normally obligatory in commercial systems.

4. To explore an unusual style of C programming that makes heavy use of pre-processing facilities.

Examples of the use of the language in a variety of topics are provided in an appendix.

We are indebted to a number of colleagues for advice and help: Anthony Howe, David Steinbrook, Bob Bernecky, Mark Czerwinski, L.J. Dickey, Jiri Dvorak, James Hui, Eric Iverson, Paul Jackson, and Roland Pesch.

A. ORTHOGRAPHY

At the time of the first implementation of APL, the then-new IBM Selectric typewriter with its changeable type element offered a welcome escape from the limitations of the existing printers, which provided only a few symbols beyond a one-case alphabet, punctuation, and the decimal digits. The Selectric was exploited by designing an alphabet that provided single-character spelling of all words in the language (except for the literal names used for variables).

This spelling scheme offered several advantages, due to the fact that the words were:

1. *Mnemonic*, using the shapes of symbols to suggest the functions denoted, as in up- and down-arrows for the functions *take* and *drop*.

2. *Universal*, in avoiding mnemonic devices rooted in particular natural languages.

3. *Disjoint* from the literals used in variable names, so as to a) avoid the introduction of *reserved* words, b) improve readability, and c) obviate required spaces around words, as in a | b instead of a mod b.

However, special alphabets pose serious display problems, and it is desirable to have a spelling scheme based on a widely available computer alphabet. We have here attempted to design a spelling scheme based on the ASCII alphabet that retains the advantages cited above for the older spelling scheme.

Words are spelled with one character or with two, the last of which is a period or a colon; words are formed by scanning from right to left, each colon or period (not in a number) combining with the character to its left to form a word. Any number of spaces may be used between words, but spaces are not required, except that in a number, a space or zero must precede a decimal point that is not preceded by a digit or negative sign.

The spelling scheme is shown in the language summary of Table 1, a study of which should clarify the application of the following guides used in its design:

1. Adopt mathematical symbols (such as + − < > ! ~), and symbols whose shape or usage somehow suggest the mathematical notions, as in the number sign # for number of items (in an argument, or selected in replication), and De Morgan's use of ∧ for power [2].

2. Use single characters for other primitives whose use should become common, as in & @ ; " _ and \ for composition, upon and defer, link, rank and under, negative sign, and scan and outer product.

3. Use a dot or colon with a common mathematical character that suggests the function, as in <. and >. for min and max, and in =. and =: for local and global assignment.

4. Use a dot with a letter that suggests a mathematical symbol or definition, as in o. for the family of circular functions, in e. for membership (because epsilon is used for it in mathematics), in i. for integers, and in x. and y. for arguments (because of the analogous use of x and y in mathematics).

5. Use related strings for related functions, as in ∧ for exponential, ∧. for its inverse (natural log), and ∧: for root and square root; in #. for base value, and #: for its inverse; in + and * for *plus* and *times*, and +. and *. for *or* and *and* (their analogs in logic); and in @ for a conjunction that permutes axes, and @. and @: for other permutations.

6. Adopt mnemonic aids such as the three cases of $ (which suggests an S) for Shape, Sequence list, and Self-reference; and .. for the *Dot* or *Inner* product. This adoption of the double dot obviates the spaces previously needed around the dot in some cases.

Anyone who is familiar with earlier spelling of APL words, or who is using earlier APL literature, may find it helpful to pronounce them in the traditional way, as in *iota* for ι..

The function /. cuts its list argument into words according to rules appropriate to an APL sentence. Thus, /. '+/3 4 5 *ι.3' yields the boxed list + and / and 3 4 5 and * and ι. and 3.

	.	**:**
= NubClassify ; Equal	Is (Local)	Is (Global)
< Box ; LessThan	Floor ; Minimum	Decrement ; LeOrEq
> Open ; GreaterThan	Ceiling ; Maximum	Increment ; GtOrEq
_ NegativeSign/Infinity		
+ Conjugate ; Plus	; GCD (Or)	; Nor
* Signum ;Times	; LCM (And)	; Nand
− Negate ; Minus	Reverse ; Rotate	; Match
% Reciprocal ; Divide	MatrixInv ; MDiv	
∧ Exponential ; Power	NaturalLog ; Log	SquareRoot ; Root
$ ShapeOf ; Shape	SequenceList	SelfReference
~ Both ; Cross	Not (1-) ; Less	Nubsieve ; NotEqual
| Magnitude ; Residue		Custom
. NOT USABLE	Det ; DotProd	
: NOT USABLE	Companion	Definition
, Ravel ;ChainItems		Itemize ; Laminate
; Table ;Link (+Box)	BoxItems ; Link	
# Tally ; Copy	Base2 ; Base	Antibase2 ; Antibase
@ Atop-At	Dir-Cyc ; Permute	AtomPermute
/ Insert ; xWay Insert	Words	GradeUp ; Sort
\ Scan ; OuterProduct	Transpose	GradeDown ; Sort
{ Catalog ; From	Nub ; Take	Right (Dex)
} Merge	Raze ; Drop	Left (Lev)
″ ConstCutRankUnder	Execute ;Execute	Format ; Format
& Composition-With		
! Factorial ; OutOf		
? Roll ; Deal		
(Open Parenthesis		
) ClosePar-Label-Cmd		
α	Alphabet	
e	; Epsilon (Member)	
ι	Integers ; IndexOf	
o	PiTimes ; Circular	
x	First Argument	
y	Last Argument	
E	; MemberOfInterval	
X	External (Foreign)	

Table 1: LANGUAGE SUMMARY

B. MAJOR CELLS, REPLICATE, RESHAPE, and OUTER PRODUCT

Because of the importance of major cells, we will adopt the terms *item* and *atom* for the major cells and the scalars. We will also adopt the symbol # for the *item count*, or *tally*; #b is 1 if b is an atom, and is otherwise equal to 0{$ b.

The dyadic case n#b is similar to the replicate function previously provided (for historical reasons) by the derived function n/; the successive atoms of n specify the number of repetitions of successive items of b to be selected. The reshape ($) is also redefined to apply to items rather than atoms; the old behaviour is obtained by ravelling the right argument.

Catenation of the items of A and B by the expression A comma-bar B is more useful than the catenation of 1-cells provided by the comma; in particular, the catenation of 1-cells can be provided by comma-bar of rank 1. Consequently, we will use the comma for catenation of items (that is, catenation along the leading axis), and drop the symbol comma-bar. For similar reasons, the / and \ will be adopted for the meanings that were assigned to /-bar and \-bar, and the latter pair of symbols will be dropped.

The table function (previously provided by the monadic case of the comma-bar) will be provided by the semi-colon, its dyadic use being assigned to the link function. Thus, a;b is defined by (<a),b, with the right argument b automatically boxed if it is open.

The expression jot.f for outer product uses (for historical reasons) a conjunction where an adverb would serve. We will adopt the dyadic case of f\ for this purpose, and the jot and the notation jot.f will be dropped.

C. USER-DEFINED VERBS, ADVERBS and CONJUNCTIONS

The conjunction denoted in the dictionary by the inverted Greek Delta will be denoted by the double colon, and the right-arrow and $ used to denote the sequence control and self-reference will be replaced by $. and $:. The forms m: :d and 1: :a and 2: :a will be otherwise adopted.

As in the dictionary, assignment provides dynamic localization; for example, the first execution of a=.g a in a function f applies g to the global value of a, but produces a local copy. Unlike the dictionary definition, the localization is strict, so that a local copy is not available to user-defined functions that are invoked in f. Global assignment is provided by the copula =:.

Strict localization provides significant advantages over the heritable localization of earlier APL, and is now practicable because of the ease of passing parameters in boxed arguments. Direct definitions are easily provided by a simple cover function employing the forms m::'' and ''::d.

D. FROM, IOTA, and BASE

The monadic case of ι. is defined like monadic iota, but extended to list arguments as follows: ι.s is (|s)$+\0,(*/|s)$1, but reversed along each axis for which the corresponding element of s is negative; the result for an empty argument is the scalar 0. For example:

```
    ι. 2 3      ι. 2 _3    ι. ''      ι. _4
    0 1 2       2 1 0        0        3 2 1 0
    3 4 5       5 4 3
```

A new monadic case of base-value is defined as the base-2 value; that is, #.v is equivalent to 2#.v. An infinite rank monadic case of anti-base is defined as (n$2)#:a, where n is the maximum of the minimum lengths required to represent the (integer) atoms of a.

E. PERMUTATIONS

The words \. and −. will be used for transposition and for leading-axis reverse and rotate, the lines in the spelling indicating the axes involved, as they did in the old symbols for these functions. Other permutations (modelled upon, and replacing, those in the dictionary called *cycle, mix,* and *mix index*) will be represented by @. and @:.

Standard Direct and Cycle Representations. If p is a permutation of the atoms of ι.n, then p is said to be a *permutation vector* of order n, and if n=#b, then p{b is a permutation of the items of b.

The expression @.p yields a list of boxed lists of the atoms of ι.#p called the *standard cycle* representation of p. If (as in the example in the dictionary) p=. 4 5 2 1 0 3, then @.p yields

2;4 0; 5 3 1 because the permutation p moves to position 2
the item 2, to 4 the item 0, to 0 the item 4, to 5 the item 3, to
3 the item 1, and to 1 the item 5. The monad @. is self-inverse;
when applied to a standard cycle representation it produces the
corresponding *direct* representation.

A given permutation could be represented by cycles in a variety
of ways, and the standard form is made unique by the following
restrictions:

> The cycles are disjoint and exhaustive (that is, the atoms of
> the boxed elements together form a permutation vector); each
> boxed cycle begins with its largest element (possible because
> any rotation of a single cycle represents the same permutation);
> and the boxed cycles are arranged in ascending order on their
> leading elements (possible because the cycles are disjoint).

Non-Standard Representations.
If d and c are direct and cycle
representations of order #*b*, then d@.*b* and c@.*b* produce the
corresponding permutations of the items of *b*. More generally,
since the item count of *b* determines the order of the permutation,
the arguments d and c may be non-standard in ways to be defined.
In particular, elements belonging to (i.2*#*b*)-#*b* are permitted,
and are treated as their residues modulo #*b*.

If q is not boxed, and if the elements of (#*b*)|q are distinct, then
q@.*b* is equivalent to d@.*b*, where d is the standard form of q
given by d=.((i.*n*)~.*n*|q),*n*|q, where *n* is #*b*. In other
words, positions occurring in q are moved to the tail end.

If q is boxed, then the elements of (#*b*)|>j{q must be distinct
for each j, and the boxes are applied in succession. For exam-
ple, (2 1;3 0 1)@.i.5 is equivalent to (<2 1)@.(<3 0
1)@.i. 5, and the result of either is the standard direct permu-
tation 1 2 3 0 4.

Atomic Representation.
If T is the table of all !*n* permutations
of order *n* arranged in lexical order (that is, /:T is i.!#T), then
k is said to be the *atomic* representation of the permutation k{T.
Moreover, k@:*b* permutes items of *b* by the permutation of order
#*b* whose atomic representation is (!#*b*)|k. For example, 1@:*b*
transposes the last two items of *b*, and _1@:*b* reverses the items,
and 3@:*b* and 4@:*b* rotate the last three items of *b*. Finally,
(i.!*n*)@:i.*n* produces the ordered table of all permutations of
order *n*, as does the *fork* [3] used in the expression (i.&!@:i.)
n.

The transformation between direct and cycle representations pro-
vided by the monad @. is extended to non-negative non-standard
cases by treating any argument q as a representation of a permu-
tation of order 1+>./}.q. Similarly, the monad @: applied to
any cycle or direct permutation yields its atomic representation.
For example, @:0 3 2 1 is 5, as are @:3 2 1 and @:0;2;3
1 and @:<3 1.

F. TRANSPOSITIONS and SECTIONS

The symbol @ will replace the *hoof*, with the noun cases of the
conjunction (*Defer* and *Prefer*) modified so that v@*n* defers axes
n of the right argument before applying v, and *n*@v defers axes
of the left. Consequently, the expression a *n*0@v@*n*1 *b* defers
axes of both arguments before applying v. The monadic cases of
v@*n* and *n*@v are identical.

If the number of elements of *n* equals the rank of v, then v@*n*
applies v to the cells selected by the axes specified by the atoms
of v, and v@*n* can therefore be said to apply v *at n*, as suggested
by the name of the symbol @.

Because {: is an identity function, transposition alone can be
obtained by using {:@*n*.

A boxed argument *n* provides sectioning, grouping the axes spec-
ified by a single box into a single result axis. For example, if *b*

has the shape i. 6 and *n*=.2;4 1;0, then the shape of {:@*n*
b is 3 5 2 1 0.

G. FORMAT

The dyadic case of format (":) is defined with both ranks 1, and
with each element e of the left argument controlling the rep-
resentation of the corresponding element of the right argument as
follows:

w=.<.|e specifies the total width allocated; if this space is
inadequate, the entire space is filled with asterisks.

d=.<.10*(|e)-w specifies the number of digits following
the decimal point (which is itself included only if d is not
zero.)

Any negative sign is placed just before the leading digit.

If e>:0, the result is right-justified in the space w

If e<0, the result is put in exponential form (with one digit be-
fore the decimal point) and is left-justified except for two fixed
spaces reserved on the left (including the one for a possible
negative sign)

The monadic rank of ": is infinite, and the result is equivalent
to the application of the dyadic definition with a left argument
chosen to provide a minimum of one space between columns.
Default output is equivalent to the use of the monadic case.

H. EXTERNAL COMMUNICATION

Communication with the keyboard, screen, and operating system
files is provided by the conjunction X., whose many arguments
provide considerable flexibility.

I. SOME IMPLICATIONS FOR TEACHING

The mere introduction of lists, scan, and outer product allows a
wealth of interesting explorations, as in +\a=.0 1 2 3 4 5
for the triangular numbers, in +\1+a+a to see that the odd num-
bers sum to squares, and in various outer products such as a+\a
and a*\a to see *addition, multiplication, remainder, divisibility*
and other tables, including the binomial coefficients (Pascal's Tri-
angle) provided by a!\a.

Lists are easily explained as the use of collective nouns, and the
scan is easily explained as an adverb. Unfortunately, the simple
and important notion of a function table required, in traditional
APL, not just a further use of an adverb, but the use of a conjunc-
tion whose first argument could only be explained as an historical
anomaly. The present use of an adverb for outer product avoids
this difficulty.

Expressions such as pr=. +% provide a simple introduction of the
notion of function definition (and of the *hook* [3]), and expres-
sions such as pr\1 2 2 2 2 2 and pr\3 7 15 1 show
interesting uses of such a defined function in producing successive
approximations to interesting quantities.

Expressions such as *sum*=.+/ and *sqrt*=.∧&0.5 and *log*=.
10&∧. and *neq*=.~.@= provide simple and interesting uses of
adverbs and conjunctions. Moreover, the general form of defini-
tion provided by the :: conjunction permits a simple introduction
to the use of iteration and recursion.

The generally useful notions of classification can be introduced by
using the outer product a<:\b in expressions for producing bar-
charts and graphs, and can be explored further using the expression
#:i.2∧*n* to produce the complete classification table of order
n. Thus if CCT=.#:i.2∧#v=.2 3 5, then v+.*CCT and
v*..∧CCT produce the sums and products over all subsets of v.

In a more specialized area, the functions @. and @: provide
powerful facilities for the discussion of permutations. Thus,

$(\iota .!4)\mathcal{Q}:\iota .4$ displays a complete table of permutations, and an expression such as $\mathcal{Q}. 4\ 3\ 0\ 1\ 2$ can provide an introduction to cycles and to the use of the LCM ($\star .$) of their lengths to determine the power of a permutation. For examples in further topics, see the appendix.

J. THE C IMPLEMENTATION

The system is implemented in C, because it is an adequate language available on a wide variety of machines. The implementation is guided by two principles: clarity, and exploitation of underlying facilities. Efficiency is not a main objective.

Clarity does not mean the micro (and relatively insignificant) clarity of individual C statements, but the macro clarity of being close to the APL or mathematical definitions. The C code is written to be understandable by an APL-knowledgeable reader.

Facilities already available in the environment are exploited: for memory management, the C library functions malloc() and free() are used, the underlying virtual memory facilities being presumed to be adequate; for session management, the system reads from standard input and writes to standard output. This, together with the ASCII spelling, makes it possible to use any of several widely-available session managers, such as EMACS or Sun-View/OpenLook.

Organization. The system is organized along the lines suggested by the dictionary, in particular, by the parser [1, p. 38]. The parsing rules are expressed in C as follows:

```
#define RHS   (NOUN+VERB+ADV+CONJ)
#define EDGE  (MARK+ASGN+LPAR)

static struct {I c[4];AF f;I b,e;}cases[] = {

EDGE+ADV+VERB,      VERB,  NOUN,  ANY,       verb,1,  2,
CONJ,               NOUN,  VERB,  NOUN,      verb,2,  3,
EDGE+ADV+VERB+NOUN, NOUN,  VERB,  NOUN,      verb,1,  3,
EDGE+ADV+VERB+NOUN, NOUN+VERB, ADV,  ANY,    adv,  1,  2,
EDGE+ADV+VERB+NOUN, NOUN+VERB, CONJ, NOUN+VERB, conj,1,  3,
EDGE+ADV+VERB+NOUN, VERB,  VERB,  VERB,      form,1,  3,
EDGE,               VERB,  VERB,  ANY,       form,1,  2,
NAME,               ASGN,  RHS,   ANY,       is,   0,  2,
LPAR,               RHS,   RPAR,  ANY,       punc,0,  2,
ANY,                ANY,   ANY,   ANY,       move,0, -1,
};
```

A sentence to be parsed is placed on a left stack, and as execution proceeds words are moved from the tail of the left stack to the front of a right stack. When the first four words of the right stack match a pattern (columns 0 to 3 of the table), the corresponding action (4) is triggered and applied to the indicated words (5, 6), with the result replacing these words.

Data Structures. The fundamental data structure is the APL array, that is, the C structure:

```
typedef long I;
typedef struct {I t,c,n,r,s[1];}*A;
```

t   type
c   reference count
n   number of atoms in the ravelled array
r   rank
s   shape list
v   atoms of the ravelled array (immediately following s)

All objects, whether numeric, literal, or boxed, whether noun, verb, adverb, conjunction, or punctuation, are represented by this structure. Most C functions in the system accept APL arrays as arguments and return them as results.

Definitions and macros. Extensive use is made of C preprocessor definitions and macros; to augment the expressive power of C, to enforce uniformity, and to increase readability. Example: An "APL function" is a function which accepts one or two APL array arguments, and returns an APL array result. The macros F1 and F2 encapsulate this convention:

```
#define F1(f) A f(w,self)A w,self;
#define F2(f) A f(a,w,self)A a,w,self;
```

(self is a pointer to function parts – rank, inverse, etc.)

A compact but readable programming style results from using such definitions. The implementation of , :y (itemize) and x, :y (laminate) are cases in point:

Itemize: , :y adds a single unit axis to y, making the shape 1, $y.

```
F1(lamin1) {R reshape(over(one,shape(w)),ravel(w));}
```

Laminate: If the shapes of x and y are equal, then x, :y is defined by (,:x),(,:y). If one is an atom a, it is first replaced by s$a, where s is the shape of the other.

```
F2(lamin2) {R over(a,reshape(over(one,shape(AR(w)?w:a))
           ,ravel(w)));}
```

Statistics. Analysis of the C implementation as it stands on 1990 2 22 yields the following statistics. (Header files and variables without functions are excluded.)

C Fns	240	Lines	1345
Lines	1345	+/ Line lengths	44722
Average lines/fn	5.6	Average chars/line	33.3
Min	1	Min	1
Max	40	Max	89
Median	1	Median	32
One-liners	125	One-character lines	91

181 of the 240 functions are APL functions.

Therefore, the implementation consists of a large number of short functions, having short lines, with a well-defined uniform interface. These are characteristic of an APL programming style.

REFERENCES

1. Iverson, K.E., A Dictionary of APL, APL Quote-Quad, Volume 18, Number 1, September 1987, pp 5-40.

2. Cajori, Florian, A History of Mathematical Notations, The Open Court Publishing Co., 1928, Volume I, Paragraph 313.

3. McDonnell, E.E., and K.E. Iverson, Phrasal Forms, APL Quote-Quad, Volume 19, Number 4, August 1989, pp 197-199.
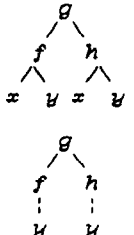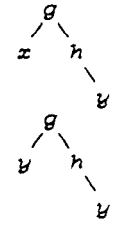
APPENDIX

The forty-five frames in the following appendix show examples of use of the system in a variety of topics. All were actually executed on the system in March 1990.

ALPHABET A

```
    $ a.
256
    j=. a. i. 'aA'
    j
97 65
    j +\ i. 9
97 98 99 100 101 102 103 104 105
65 66 67  68  69  70  71  72  73
    (j+\i.30){a.
abcdefghijklmmopqrstuvwxyz{|}~
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^
    a.{~j+\i.30
abcdefghijklmmopqrstuvwxyz{|}~
ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^
    3 + 125 * 6 % 100
10.5
    i. 2 5
0 1 2 3 4
5 6 7 8 9
    *\~0j1 _1 0j_1 1
    _1  0j_1   1    0j1
 0j_1    1    0j1    _1
    1   0j1   _1  0j_1
 0j1    _1  0j_1    1
```

SPELLING B

```
sentence=.'index=. a.i.''aA'''

    /.sentence
```

index	=.	a.	i.	'aA'

```
    $ /.sentence
5
    >/.sentence
index
=.
a.
i.
'aA'
    ". sentence
97 65

    ". 'abc =. 3 1 4 2'
3 1 4 2
abc
3 1 4 2
```

GRAMMAR C

```
    fahrenheit =. 50
    (fahrenheit - 32) * 5 % 9
10
    prices =. 3 1 4 2
    orders =. 2 0 2 1
    orders * prices
6 0 8 2
    +/ orders * prices
16
    +\ 1 2 3 4 5
1 3 6 10 15
    2 3 * \ 1 2 3 4 5
2 4 6  8 10
3 6 9 12 15
    decr=. - & 1
    decr _1 0 1 2 3
_2 _1 0 1 2
```

PARTS OF SPEECH

50 fahrenheit	Nouns/Pronouns
+ - * % decr	Verbs/Proverbs
/ \	Adverbs
&	Conjunction
=.	Verb-to-be
( )	Punctuation

TABLES Da

```
    prices=. 3 1 4 2
    orders=. 2 0 2 1

    prices * orders
6 0 8 2
    prices *\ orders
6 0 6 3
2 0 2 1
8 0 8 4
4 0 4 2

    TO READ A TABLE,
    BORDER IT BY ITS
    ARGUMENTS:

     *  | 2 0 2 1
    -----------
     3  | 6 0 6 3
     1  | 2 0 2 1
     4  | 8 0 8 4
     2  | 4 0 4 2
```

TABLES Db

```
    n=. 0 1 2 3
    n +\ n
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6

    *\ ~ n
0 0 0 0
0 1 2 3
0 2 4 6
0 3 6 9

    ^\ ~ i. 4
1 0 0  0
1 1 1  1
1 2 4  8
1 3 9 27
    +.\~ 0 1
0 1
1 1
    +:\~ 0 1
1 0
0 0
```

TABLES Dc

```
    |\ ~ 1+i. 5
0 0 0 0 0
1 0 1 0 1
1 2 0 1 2
1 2 3 0 1
1 2 3 4 0

    +/ 0= |\ ~ j=. 1+i. 15
1 2 2 3 2 4 2 4 3 4 2 6 2 4 4
    2=+/0=|\ ~ j
0 1 1 0 1 0 1 0 0 0 1 0 1 0 0
    (2=+/0=|\~ j) # j
2 3 5 7 11 13

    =\~i. 4
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
    <:\~i. 4
1 1 1 1
0 1 1 1
0 0 1 1
0 0 0 1
```

TABLES Dd

```
    text=. ' i sing of olaf '
    text=. text,'glad and big'

    alph=. ' abcdefghijklmmo'
    alph=. alph,'pqrstuvwxyz'

    '01'{~10{.alph=\text
1010000100100001000010001000
0000000000000010001001000000
0000000000000100000000000100
0000000000000000000000000000
0000000000000000100010000000
0000000000000000000000000000
0000000010000100000000000000
0000001000000100000000000001
0000000000000000000000000000
0100100000000000000000000010

    2 13$+/"1 alph=\text
7 3 1 0 2 0 2 3 0 3 0 0 2
0 2 2 0 0 0 1 0 0 0 0 0 0
```

CLASSIFICATION Ea

```
    x=. 1 2 3 4 5 6 7
    y=. (x-3) * (x-5)
    y
8 3 0 _1 0 3 8

    range=. m-i. 1+(m=. >./y)-<./y
    range
8 7 6 5 4 3 2 1 0 _1

    bc=. range <:\ y
    bc
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 1 0 0 0 1 1
1 1 0 0 0 1 1
1 1 0 0 0 1 1
1 1 1 0 1 1 1
1 1 1 1 1 1 1
```

CLASSIFICATION Eb

```
    x=. 1 2 3 4 5 6 7
    y=. (x-3) * (x-5)
    y
8 3 0 _1 0 3 8

    range=. m-i. >:(m=. >./y)-<./y
    range
8 7 6 5 4 3 2 1 0 _1

    bc=. range <:\ y
    bc { ' *'
*     *
*     *
*     *
*     *
*     *
**   **
**   **
**   **
*** ***
******

    <\ 0 0 0 1 0 1 1 0 1
0 0 0 1 0 0 0 0 0
```

```
     CLASSIFICATION: graphs Ec
     bc
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 0 0 0 0 0 1
1 1 0 0 0 1 1
1 1 0 0 0 1 1
1 1 0 0 0 1 1
1 1 1 0 1 1 1
1 1 1 1 1 1 1

     <\ bc
1 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 0 0
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 0 1 0 0
0 0 0 1 0 0 0
```

```
     CLASSIFICATION: graphs Ed
   <\bc
1 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 0 0
0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 0 1 0 0
0 0 0 1 0 0 0

   ' *' {~ <\bc
 *       *



 *     *

 * *
  *
```

```
     CLASSIFICATION +,* on subsets Ee
a=. 0 0 0 0 1 1 1 1
b=. 0 0 1 1 0 0 1 1
c=. 0 1 0 1 0 1 0 1

cct=. a,b,,:c
cct
0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1

   2 3 5 +..* cct
0 5 3 8 2 7 5 10
   2 3 5 *..^ cct
1 5 3 15 2 10 6 30

   +/cct
0 1 1 2 1 2 2 3
   {:c2=.(2=+/cct)#"1 cct
0 1 1
1 0 1
1 1 0
   2 3 5 >. .. * c2
5 5 3
```

```
        CLASSIFICATION dot products Ef

   2 3 5 | 0 5 3 8 2 7 5 10
   4 2 1 | 0 1 2 3 4 5 6 7
   ------+------------------
         | 0 0 0 0 1 1 1 1
         | 0 0 1 1 0 0 1 1
         | 0 1 0 1 0 1 0 1

   row0 =. 2 3 5 }: col5=. 1 0 1

   row0 * col5
2 0 5
   +/ row0 * col5
7

   row0 +..* cct
0 5 3 8 2 7 5 10
   row0 *..^ cct
1 5 3 15 2 10 6 30

   row0 ^ col5
2 1 5
   */ row0 ^ col5
10
```

```
                    STRUCTURES: box Fa
   text
i sing of olaf glad and big

   -. text
gib dna dalg falo fo gnis i
   < 'glad'
+----+
|glad|
+----+
   u=. (<'glad'),(<'and'),<'big'
   u
+----+---+---+
|glad|and|big|
+----+---+---+
   -. u
+---+---+----+
|big|and|glad|
+---+---+----+
   # u
3
   'glad';'and';'big'
+----+---+---+
|glad|and|big|
+----+---+---+
```

```
                    STRUCTURES: each Fb
   text
i sing of olaf glad and big

   words=. /. text

   words
+-+----+--+----+----+---+---+
|i|sing|of|olaf|glad|and|big|
+-+----+--+----+----+---+---+
   1 0 2 3 { words
+----+-+--+----+
|sing|i|of|olaf|
+----+-+--+----+
   -. " > 1 0 2 3 { words
+----+-+--+----+
|gnis|i|fo|falo|
+----+-+--+----+
   _1"< text
+-+----+--+----+----+---+---+
|i|sing|of|olaf|glad|and|big|
+-+----+--+----+----+---+---+
```

```
             STRUCTURES open Fc
   words
+-+----+--+----+----+---+---+
|i|sing|of|olaf|glad|and|big|
+-+----+--+----+----+---+---+

   tt=. > words
   tt
i
sing
of
olaf
glad
and
big

   $ tt
7 4
```

```
             PROGRAMS: simple Ga
   root=. 'y. ^ %2'::'y. ^ %x.'
   root 64
8
   3 root 64
4
   rPr=. '% y.'::'x. + % y.'
   3 rPr 4
3.25
   rPr / 1 2 2 2 2 2 2
1.4142
   rPr \ 1 2 2 2 2
1 1.5 1.4 1.41667 1.41379
   rPr \ 3 7 15
3 3.14286 3.14151
   triple=. '3*y.'::''
   triple i.5
0 3 6 9 12
   3 triple 6
domain error
   tr=. '3*y.'::*
   tr i. 5
0 3 6 9 12
   3 5 7 tr i. 3
0 5 14
```

```
             PROGRAMS: conditional Gb
   p=. '$.=. 1+y.<0'
   q=. 'y. ^ %2'
   r=. '''DOMAIN ERROR'''

   conditional=. (p;q;r)::''

   conditional _49
DOMAIN ERROR

   conditional 49
7

   tozero=.(p;'y.-1';'y.+1')::''

   tozero 3
2
   tozero _3
_2
   tozero "0 (_2 _1 0 1 2 3)
_1 0 _1 0 1 2
```

```
            PROGRAMS: iterative Gc
   a=. 'r=. 1 }: $.=. y. # 1'
   b=. 'r=. r * 1+ # $.'

   factorial=. (a;b)::''
   factorial 5
120
   factorial"0 i. 6
1 1 2 6 24 120

   > a;b
r=. 1 }: $.=. y. # 1
r=. r * 1+ # $.
   c=. 'r=. (0,r) + (r,0)'
   binomials=. (a;c)::''
   binomials 4
1 4 6 4 1
   fib=.(a;'r=.r,+/(_2){.r')::''
   fib 10
1 1 2 3 5 8 13 21 34 55 89
   d=. 'r=. 1 }: $.=. x. # 1'
   e=. 'r=. (r*1+y.=.y.-1)%1+#$.'
   outof=. ''::(d;e)
   3 outof 5
10
```

```
            PROGRAMS: recursive Gd
   a=. '$.=. 2-0=y.' ; '1'
   b=. 'y. * $: y.-1'
   factorial=. (a,<b)::''
   factorial 5
120

   d=. '(r,0)+0,r=. $: y.-1'
   binomial=. (a,<d)::''
   binomial 4
1 4 6 4 1

   f=. 'r,+/(_2){.r=. $: y.-1'
   fibonacci=. (a,<f)::''
   fibonacci 10
1 1 2 3 5 8 13 21 34 55 89

   g=. '$.=. 2-0=x.' ; '1'
   h=. 'y.*x.%~x.$:d<:y.'
   outof=. ''::(g,<h)
   outof"0\~i. 4
1 1 1 1
0 1 2 3
0 0 1 3
0 0 0 1
```

```
            PROGRAMS: recursive Ge
   a=.'$.=.1+0<n=.x.-1'
   b=.',:2{.y.'
   c=.'(n$:0 2 1{y.),(1$:y.),'
   hanoi=.''::(a;b;c,'n$:-.y.')

   2 hanoi 'ABC'
AC
AB
CB
   \. 4 hanoi 0 1 2
0 0 2 0 1 1 0 0 2 2 1 2 0 0 2
2 1 1 2 0 2 2 1 1 0 0 1 2 1 1

   \. 'ABC'{~ 4 hanoi 0 1 2
AACABBAACCBCAAC
CBBCACCBBAABCBB

   c=.'r=.0#$.=.y.#1+n=.0'
   d=.'r=.r,(n=.1+n),r'
   h=.(c;d)::''
   h 4
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
   h 3
1 2 1 3 1 2 1
```

```
            PROGRAMS: recursive Gf
   {:a=.3 3$'abcdefghi'
abc
def
ghi
   (f=.f~."1 0 f=.i.&#) a
1 2
0 2
0 1
   <"2 (minors=.f { 1&}."1) a
```

ef	bc	bc
hi	hi	ef

```
   p=.'$.=. 1+1=#y.' }: r=.'0{,y.'
   q=.'(0{"1 y.)-..*$:"2 minors y.'
   {:b=.?3 3$9
1 6 4
4 1 0
6 6 8
   (det=.(p;q;r)::'') b
_112
   s=.'(0{"1 y.)+..*$:"2 minors y.'
   (permanent=.(p;s;r)::'') b
320
```

```
            GEOMETRY: 2-space Ha
   length=. '^:+/y.^2'::''
   length 12 5
13
   {: tri=. ? 2 3 $ 9
3 4 7
0 0 4
   1 -."1 tri
4 7 3
0 4 0
   {:lsides=.length tri-1-."1 tri
1 5 5.65685
   {: semiper=. 2 %~ +/lsides
5.82843
   area=. ^:*/semiper-0,lsides
   area
2
   tri,1
3 4 7
0 0 4
1 1 1

   2 %~ det tri,1
2
```

```
            GEOMETRY: 3-space Hb
   tri,1
3 4 7
0 0 4
1 1 1
   2%~ det tri,1
2
   2%~ det 1 0 2 {"1 tri,1
_2
   {: tetrahedron=. 0,"1 =\~ i. 3
0 1 0 0
0 0 1 0
0 0 0 1

   volume=. det&(,&1) % !&#

   volume tetrahedron
_0.166667

   {: tet=. ? 3 4 $ 9
6 0 3 0
3 6 5 8
7 4 0 5
   volume tet
11.5
```

```
            CONNECTIONS: arcs Ia

   arcs=. ? 22 2 $ 8 8
   8 {. arcs
1 6
3 4
1 0
5 5
7 3
4 6
0 0
4 5

   \. n=.arcs{nodes=. 'ABCDEFGH'
BDBFHEAEAAFHEFFGACGCHG
GEAFDGAFDDEGADHCFFHBFF

   6{. barcs=. <"1 n
```

BG	DE	BA	FF	HD	EG

```
   15 {. ,arcs
1 6 3 4 1 0 5 5 7 3 4 6 0 0 4
```

```
            CONNECTIONS: conn. matrix Ib

   '01234567' {~ \. arcs
1315740400574556026276
6405360533460372557155
   b=. '(i.,~x.)e.'
   cmFarcs=. ''::(b,'y.+..*x.,,1')

   cm=. 8 cmFarcs arcs
   cm
1 0 0 1 0 1 0 0
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 0 0 1 0 0 0
1 0 0 0 0 1 1 0
0 0 0 1 1 1 0 1
0 0 1 0 0 1 0 1
0 0 0 1 0 1 1 0

   +/cm
3 1 1 3 2 6 3 2
   +/+/cm
21
```

```
            CONNECTIONS: family Ic
   cm
1 0 0 1 0 1 0 0
1 0 0 0 0 0 1 0
0 1 0 0 0 1 0 0
0 0 0 0 1 0 0 0
1 0 0 0 0 1 1 0
0 0 0 1 1 1 0 1
0 0 1 0 0 1 0 1
0 0 0 1 0 1 1 0

   points=. 1 0 0 0 0 0 0 1

   points +. .. *. cm
1 0 0 1 0 1 1 0

   points+.points+. .. *.cm
1 0 0 1 0 1 1 1

   immfam=. ''::'x.+.x.+...*.y.'
   points immfam cm
1 0 0 1 0 1 1 1
   fam=.''::'immfam&y...(#y.)x.'
   points fam cm
1 1 1 1 1 1 1 1
```

```
{: cm2=. 0=78 8 $ 5
1 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 0 0 1 0

  points=. 1 0 0 0 0 0 0 1
  points fam cm2
1 0 1 1 0 0 1 1

  cm2 fam cm2
1 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
1 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 1 0 0 1 0
```

```
a=. 0 0 0 0 1 1 1 1
b=. 0 0 1 1 0 0 1 1
c=. 0 1 0 1 0 1 0 1
$d=. a,b,,:c
3 8
 adj=. '1=y. +..~: \. y.'::''
{. e=. adj \. d
0 1 1 0 1 0 0 0
1 0 0 1 0 1 0 0
1 0 0 1 0 0 1 0
0 1 1 0 0 0 0 1
1 0 0 0 0 1 1 0
0 1 0 0 1 0 0 1
0 0 1 0 1 0 0 1
0 0 0 1 0 1 1 0
  e{' *'
** *
*  * *
*  *  *
**     *
*     **
*  *  *
*  *  *
*  **
```

```
t=.'i sing of olaf glad and big'
  {: tt=. > /. t
i
sing
of
olaf
glad
and
big

  /: tt
5 6 4 0 2 3 1

  tt /: tt
and
big
glad
i
of
olaf
sing
```

```
o=.'(' }: c=.')' }: s=.'-'
minus=. ''::'o,x.,c,s,y.'
'a' minus 'b'
(a)-b
list=.'defg'
minus / list
(d)-(e)-(f)-g
minus\list
d
(d)-e
(d)-(e)-f
(d)-(e)-(f)-g
d,e,f,g=.<:f=.<:e=.<:d=.4
4 3 2 1
". minus / list
2
". minus \ list
4 1 3 2
times=. ''::'o,x.,c,''*'',y.'
list times".0 -. list
(d)*g
(e)*f
(f)*e
(g)*d
```

```
^&2 c=. 1+i.4
1 4 9 16
2&^ c
2 4 8 16
pow=. ^&2::^
pow c pow c
1 16 729 65536
c +&% c
2 1 0.666667 0.5
tr=. 3&*::*
db=. 2&*::*
c tr & db c
4 16 36 64
c db & tr c
9 36 81 144
c +&^. c
0 1.38629 2.19722 2.77259
^ c +&^. c
1 4 9 16
db & tr \~ c
 9 18  27  36
18 36  54  72
27 54  81 108
36 72 108 144
```

```
c |@- \ c=. 1+ i. 4
0 1 2 3
1 0 1 2
2 1 0 1
3 2 1 0

db=. 2&*::*
tr=. 3&*::*

db @ tr \ ~ c
2  4  6  8
4  8 12 16
6 12 18 24
8 16 24 32
db & tr \ ~ c
 9 18  27  36
18 36  54  72
27 54  81 108
36 72 108 144
```

```
+.\ a=. 0 0 1 0 1 0 1 1 0 0 0
0 0 1 1 1 1 1 1 1 1 1
+.\ -. a
0 0 0 1 1 1 1 1 1 1
-. +.\ -. a
1 1 1 1 1 1 1 0 0 0
+.\ " -. a
1 1 1 1 1 1 1 0 0 0
b=. 1 2 3 4 }: c=. 3 4 5 6
b +&^. c
1.09861 2.07944 2.70805 3.17805
^ b +&^. c
3 8 15 24
b +"^. c
3 8 15 24
{:text=.'i';'sing';'of';'olaf'
```

i	sing	of	olaf

```
-."> text
```

i	gnis	fo	falo

```
c(+ * -)d=.-.c=. i. 4
_9 _3 3 9
q=.+*-
c q \ c
0 _1 _4 _9
1  0 _3 _8
4  3  0 _5
9  8  5  0
q c
0 _1 _4 _9
r=. -,+
c r d
_3 3
_1 3
1 3
3 3
db=. 2&*::*
tr=. 3&*::*
(db+tr) c
0 5 10 15
(db*tr) c
0 6 24 54
(db*db+tr) c
0 10 40 90
```

```
         g
        / \
       f   h
      /\   /\
     x  y x  y

         g
        / \
       f   h
       :   :
       y   y
```

```
a=.5 6 7 8
b=.1 2 3 4
(*>:) b
2 6 12 20
a (*>:) b
10 18 28 40

a (*>:)\ b
10 15 20 25
12 18 24 30
14 21 28 35
16 24 32 40
(+%)/ 1 2 2 2 2 2 2
1.4142
(+%)\ 1 2 2 2 2
1 1.5 1.4 1.41667 1.41379
(+%)\ 3 7 15
3 3.14286 3.14151
(+%)\ 1 1 1 1 1
1 2 1.5 1.66667 1.6
(-%)\ 1 2 2 2 2 2
1 0.5 0.333333 0.25 0.2 0.166667
*~ (+%)/ 1 , 12 $ 1 2
3
```

```
         g
        / \
       x   h
            \
             y

         g
        / \
       y   h
            \
             y
```

FUNCTIONAL PROGRAMMING Ma

```
bc=. 0&, + ,&0
bc 1
1 1
bc bc 1
1 2 1
bc bc bc 1
1 3 3 1
q=. 'S.=.1,y.#2'
r=. 'f=. {:' ; 'f=. x.&f'
power=. 2::(q;r)
bc power 3 (1)
1 3 3 1
bc .. 3 (1)
1 3 3 1
c3=. (0&,+,&0) .. 3
c3 1
1 3 3 1
2&* .. 3"0 i. 5
0 8 16 24 32
2&+ .. 3"0 i. 5
6 7 8 9 10
g=. *~::-
5 g g 4
_11
```

SETS: propositions Na

```
{: a=. 2%~ i. 11
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
(2&<: *. <&5) a
0 0 0 0 1 1 1 1 1 0
((2&<: *. <&5) a) # a
2 2.5 3 3.5 4 4.5
((2&<: *. <&5) # {:) a
2 2.5 3 3.5 4 4.5
({: #~ 2&<: *. <&5) a
2 2.5 3 3.5 4 4.5
int=. = <.
int a
1 0 1 0 1 0 1 0 1 0 1
((2&<: *. int) a) # a
2 3 4 5
({: #~ 2&<: *. int) a
2 3 4 5
(#~ 2&<: *. int) a
2 3 4 5
```

SETS: relations Nb

```
i=.i.8  }: p=. 2 3 5 7 11
belongsto=. +./"1 @ (=\)
i belongsto p
0 0 1 1 0 1 0 1
e=. belongsto
p e i
1 1 1 1 0
c=. ~.@v=. e&'aeiou'
alph=. 'abcdefghijklmno'
alph=. alph,'pqrstuvwxyz'
(v alph)#alph
aeiou
(#~ c) alph
bcdfghjklmnpqrstvwxyz
```

SETS: union, etc. Nc

```
(even=. 0&=&(2&|))a=. i. 16
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
prime=.'2=+/0=y.|~1+i.y.'::''"0
prime a
0 0 1 1 0 1 0 1 0 0 0 1 0 1 0 0
(prime a) # a
2 3 5 7 11 13
a#~(prime*.even)a     [SET INTER-
2                          SECTION]
a#~(prime>even)a      [SET
3 5 7 11 13             DIFFERENCE]
triple=.0&=&(3&|)
q=. even+.triple     [SET UNION]
(q a) # a
0 2 3 4 6 8 9 10 12 14 15
r=. prime +. even *. triple
(r a) # a
0 2 3 5 6 7 11 12 13
```

FAMILIES OF FUNCTIONS Oa

```
x=.1 2 3 4 5 6 7
x^2
1 4 9 16 25 36 49
x^3
1 8 27 64 125 216 343
(4*x^2) + (_3*x^3)
1 _8 _45 _128 _275 _504 _833

2 3 ^~\ x
1 4  9 16  25  36  49
1 8 27 64 125 216 343
4 _3 +..*2 3 ^~\x
1 _8 _45 _128 _275 _504 _833


e=. 0 1 2 3 4
vandermonde=. e ^~\ x
vandermonde
1  1  1   1   1    1    1
1  2  3   4   5    6    7
1  4  9  16  25   36   49
1  8 27  64 125  216  343
1 16 81 256 625 1296 2401
```

FAMILIES OF FUNCTIONS Ob

```
c=. 4 2 _3 2 1
vandermonde
1  1  1   1   1    1    1
1  2  3   4   5    6    7
1  4  9  16  25   36   49
1  8 27  64 125  216  343
1 16 81 256 625 1296 2401
c+..*vandermonde
6 28 118 348 814 1636 2958

poly=. ''::'x.+..*\.y.^\i.#x.'
c poly x
6 28 118 348 814 1636 2958
```

INVERSES AND DUALITY Pa

```
cFf=.    '(y.-32) * 5%9'::''
fFc=. '32 + (y. * 9%5)'::''
dc=. 40 -~ 20 * i. 8
fFc dc
_40 _4 32 68 104 140 176 212
cFf fFc dc
_40 _20 0 20 40 60 80 100
% % 1 2 3
1 2 3
log =.  '10 ^.y.'::''
invlog=. '10 ^ y.'::''
log y=. 24 4 75 7
1.38021 0.60206 1.87506 0.845098
+/ log y
4.70243
invlog +/ log y
50400
```

INVERSES AND DUALITY Pb

```
r=. 2 3 4  }: s=. 2 4 5

invlog (log r) + (log s)
4 12 20
r * s
4 12 20

^. (^. r) + (^. s)
4 12 20
r +"^. s
4 12 20

r +"% s
1 1.71429 2.22222
% (%r) + (%s)
1 1.71429 2.22222

+"% / r
0.923077
% +/ % r
0.923077
```

INVERSES AND DUALITY Pc

```
f=. +&3
g=. -&3

{: x=. i. 4
0 1 2 3

f x
3 4 5 6

!f x
6 24 120 720

g!f x
3 21 117 717

!"f x
3 21 117 717

!"(+&3) x
3 21 117 717

!"(*&2) x
0.5 1 12 360
```