SPECIAL FEATURE:

J

BY KENNETH E. IVERSON

J is a dialect of APL first presented at APL90 [1]. It is designed for easy porting, and is already available as freeware on a number of machines [2]. The present paper is an informal introduction to J, designed to illustrate its innovations, as well as its potential use in teaching. It begins with examples, and concludes with a discussion of its relationship with other dialects of APL. The reader is advised to study each group of examples, and perhaps to experiment with them and related expressions on a J system, before reading the accompanying comments.

J

by Kenneth E. Iverson

Acknowledgements

I am indebted to a number of colleagues: to Roger Hui for collaboration in designing and implementing J, and to E.E. McDonnell and Roger Hui for suggestions used in the examples. I am also grateful to David Ziemann for the invitation to contribute to the conference issue of VECTOR.

```
   i. 4
0 1 2 3
   a=. i. 4
   2*a
0 2 4 6
   +/a
6
   a+/a
0 1 2 3
1 2 3 4
2 3 4 5
3 4 5 6
   a!/a
1 1 1 1
0 1 2 3
0 0 1 3
0 0 0 1
   a-/a
0 _1 _2 _3
1  0 _1 _2
2  1  0 _1
3  2  1  0
   ;: 'abc=.i.4 5'
```

abc	=.	i.	4 5

SPELLING. The words used to denote primitives in J are spelled using ASCII characters, and are formed from one ASCII character or from two, the last of which is either a dot or a colon. Such words need not be surrounded by spaces.

In an expression such as +/ the adverb / applies to the verb + to form a related verb whose monadic case is called *reduction* (or *insertion*) and whose dyadic case is called *outer product* (or *function table*).

The verb ;: boxes the words in its argument according to the rhematic rules of J.

```
   (<'abc'),(<'de')
```

abc	de

```
   'abc';'de'
```

abc	de

```
   <\'abcd'
```

a	ab	abc	abcd

```
   <\0 1 2 3
```

0	0 1	0 1 2	0 1 2 3

```
   +/\0 1 2 3
0 1 3 6
   +/\. 0 1 2 3
6 6 5 3
```

BOX and SCAN. The monad (monadic case of) < boxes its argument to form an atomic representation of it (of rank 0).

The prefix scan adverb \ applies its verb argument (in these examples the monads < and +/) to each prefix of the argument.

The suffix scan \. applies its verb argument to each suffix.

```
   a=.1 2 3 4 5
   sum=. +/
   sum a
15
   prefix=. \
   sum prefix a
1 3 6 10 15
   div=. %
   3 div 5
0.6
   and=. *.
   or =. +.
   nor=. +:
   not=. -.
   min=. <.
   floor=. <.
   power=. ^
   is=. =.
   log is ^.
   (a=2) or (a=4)
0 1 0 1 0
```

IS. The copula =. may be used to assign names to entities other than nouns.

In particular it can be used to assign any desired mnemonic to a primitive. Assign suitable names to primitives chosen from the Language Summary (Appendix A), using experimentation (or the names from the Summary) as guides to the behaviour of the primitives.

```
   16^0.5
4
   0.5^~16
4
   ^~1 2 3 4
1 4 27 256
   into=.%~
   6 into sum i. 6
2.5
   randompermutation=. ?~
   randompermutation 6
2 4 1 3 0 5
   2^5
32
   2&^5
32
   poweroftwo=. 2&^
   sqrt=. ^& 0.5
   sqrt 0 1 2 3 4
0 1 1.4142 1.7321 2

   t=. i. 2 3 4
   t
 0  1  2  3
 4  5  6  7
 8  9 10 11

12 13 14 15
16 17 18 19
20 21 22 23
   ,"2 t
 0  1  2  3  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 23
   +/t
12 14 16 18
20 22 24 26
28 30 32 34
   +/"2 t
12 15 18 21
48 51 54 57
   +/"1 t
 6 22 38
54 70 86
```

CROSS and COMPOSITION. The adverb ~ *commutes* or *crosses* connections to its argument verb. When *f*~ is applied monadically (as in *?~y*), it uses *y* as both arguments to the dyad *f*. Try `/: 'cat'` and `/:~'cat'`.

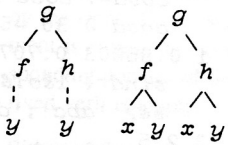
The *and* conjunction `&` attaches its noun argument to a verb to produce the corresponding monad. Experiment with cases such as `10&^.` (base-ten log), and `e.&'aeiou'` (vowel-test); apply it to an argument such as `'i sing of olaf'`

RANK. The rank conjunction " applies its left argument to cells of rank determined by its right argument.

Insertion (denoted by /) applies its left argument between the items (major cells) of the argument of the derived verb. The rank operator applied to +/ produces summation over other axes.

```
   dts=. -*+
   7 dts 4
 33
   7 (-*+) 4
 33
   3!5
 10
   3(!*-)5
_20
   (!*-)5
_600
   (!,-)5
 120 _5
   (!,-) 3 4 5
  6 _3
 24 _4
120 _5
   2 (> or =) i. 5
1 1 1 0 0
   (>&2 and <&5) i. 7
0 0 0 1 1 0 0
   t=. i. 3 4
   t
0 1  2  3
4 5  6  7
8 9 10 11
   # b=. 3 2 5 4 1
5
   #t
3
   3 %: 27 64 125
3 4 5
   am=. # %~ +/
   am b
3
   gm=. # %: */
   gm b
2.6052
   am t
4 5 6 7
   am"1 t
1.5 5.5 9.5
```

FORK. Three successive verbs (in isolation) comprise a *fork*. The monadic and dyadic cases of the fork (*fgh*) are defined by the following diagrams:



yields the number of items (major cells) of its argument, and *am* is therefore the arithmetic mean of the items of its argument.

Since *n%:y* denotes the n-th root of *y*, *gm* is the geometric mean.

The rank conjunction serves to apply *am* individually to lesser cells.

Experiment with expressions such as (^*!-%) 1+*i.* 4 to illustrate that an expression of the form (*defgh*) is equivalent to the fork (*dez*), where *z=.fgh*.

Experiment with the analogous forms (*p or q and r*) for propositions *p*, *q*, and *r*.

Examine the tautology *taut=.* <: = < *or* =

Apply the geometric mean *gm* to the argument *i.* 2 3 4. Also try *gm*"2, and *gm*"1.

```
   cos=. 2&o.
   g=. %&180
   rfd=. o.&g
   rfd 0 30 45 60
0 0.5236 0.7854 1.0472
   cosd=. cos& rfd
   cosd 0 30 45 60
1 0.86603 0.70711 0.5
   sind=. 1&o.&(o.&(%&180))
   #&> 'abc';'de';'fghij'
3 2 5
```

COMPOSITION of VERBS. *rfd* gives radians from degrees, and *cos&rfd* is the composition of the functions *cos* and *rfd*. Hence *cosd* is the cosine function for degree arguments.

```
   3 (+%) 5
3.2
   (+%)/10#1
1.6182
   (+%)/\7#1
1 2 1.5 1.6667 1.6 1.625 1.6154
   (+%)/1 2 2 2 2 2 2
1.4142
   (+%)/\3 7 15 1
3 3.1429 3.1415 3.1416
   integertest=. =<.
   integertest 2%~i. 8
1 0 1 0 1 0 1 0
```

HOOK. Two verbs (in isolation) comprise a *hook*. The monadic and dyadic cases are defined by the following diagrams:

```
 g     g
/ \   / \
y  h  x  h
   :     :
   y     y
```

The reductions by the hook +% provide continued fraction [3] approximations to the golden mean, the square-root of 2, and pi.

```
   >: y=. 0 1 2 3
1 2 3 4
   >:^:2 y
2 3 4 5
   >:^: _1 0 1 2 3 y
_1 0 1 2 3
 0 1 2 3 4
 1 2 3 4 5
 2 3 4 5 6
   0+^:7 (1)
13
   0+^:(i.9) 1
0 1 1 2 3 5 8 13 21
```

POWER and CHAINS. The conjunction ^: produces a verb whose monadic case *v*^:*k y* is defined as *k* applications of the verb *v* to the argument *y*, and whose dyadic case *x v* ^:*k y* is defined as the last of a chain of *k*+1 values beginning with *t*0=.*x* and *t*1=.*y* and *t*2=.*t*0 *v t*1 and *t*3=.*t*1 *v t*2, etc.

```
   root =. 'y.^0.5' :: 'y.^%x.'
   root a=. 0 1 64 729 4096
0 1 8 27 64
   3 root a
0 1 4 9 16
   r=. 'y.^0.5' :: ''
   3 r a
domain error
   (log=. '10^.y.' :: ^.) 1e6
6
   100 log 1e6
3
   m1=.'$.=. 1,y.#2'
   m=. m1;'t=. 1';'t=. t*1+#$.'
   >m
$.=. 1,y.#2
t=. 1
t=. t*1+#$.
   factorial=. m::''
   factorial"0 i. 6
1 1 2 6 24 120
   M1=. '$.=. 2-y.=0'
   fact=. (M1;'1';'y.*$: y.-1')::''
   fact"0 i. 6
1 1 2 6 24 120
   power=.2::(m1;'t=.{:';'t=.x.&t')
   ! power 0 " 0 i. 6
0 1 2 3 4 5
   ! power 1 " 0 i. 6
1 1 2 6 24 120
   ! power 2"0 i. 6
1 1 2 720 6.2045e23 6.6895e198
   each=. 1::'<@x.&>'
   words=. 'i';'sing';'of';'olaf'
   ;. each words
```

i	gnis	fo	falo

```
   EACH=. 1::'x.">'
```

GENERAL DEFINITION. The conjunction :: produces a verb whose monadic case is determined by the left argument, and whose dyadic case is determined by the right. If one argument is empty, the corresponding domain of the derived verb is empty.

Either of the arguments may themselves be verbs.

The sentences in a boxed list are executed in the order specified by $. (the sequence list), which is initially set to i. ns, where ns is the number of sentences in the list.

The verb *factorial* is defined iteratively and the verb *fact* recursively, using $: (self-reference) to refer to the verb being defined.

A conjunction is defined by :: with a numeric left argument (2), and an adverb is defined by a left argument of 1.

EACH is a definition of *each* based upon the conjunction UNDER (which see).

```
   a=. i. 3 2 4
   1{a
 8  9 10 11
12 13 14 15
   0{1{a
8 9 10 11
   1 {"2 a
 4  5  6  7
12 13 14 15
20 21 22 23
   {:p=. 4 4 $ 1 3 2 0
1 3 2 0
1 3 2 0
1 3 2 0
1 3 2 0
   {/\p
1 3 2 0
3 0 2 1
0 1 2 3
1 3 2 0
   {:b=.4 5$'abcdefghijklmnopqrst'
abcde
fghij
klmno
pqrst
   i=. 2 2 $ 3 17 2 14
   i{,b
dr
co
   W=.2 2$'DRCO'
   W i}b
abCDe
fghij
klmnO
pqRst
   {'AB';'abc'
```

Aa	Ab	Ac
Ba	Bb	Bc

FROM, MERGE, and CATALOGUE. The indexing function *from* ({) selects *items* from its right argument, and can be used with rank to select items from lesser cells.

The indices i in the merge expression $s\ i\}t$ (which uses the merge adverb }) refer to positions in the ravel of t, that is to elements of $,i.\$t$. The indicated positions are replaced by elements from s.

Used with a verb argument (as in $s\ v\}t$), the indices are determined by applying the verb v to the index array $i.\$t$.

Also try $m\}y$, the monadic case of merge. If y has n items, if the elements of m belong to $i.n$, and if m has the same shape as an item of y, then the result has the same shape as m, and contains elements selected from the various items of y.

```
   a=. 100 }: b=. 1000
   log=. 10&^.
   log a,b
2 3
   a+"log b
100000
   a*b
100000
   invlog=. log ^: _1
   invlog log a,b
100 1000
   invlog (log a) + (log b)
100000
   {:c=. 0=?11#4
0 0 1 0 0 1 0 1 0 0 0
   </\c
0 0 1 0 0 0 0 0 0 0 0
   </\"¦. c
0 0 0 0 0 0 0 1 0 0 0
   a=.' abcdefghijklmnopqrstuvwxyz'
   text=.'i sing of olaf'
   {: j=. (nfa=. a&i.) text
9 0 19 9 14 7 0 15 6 0 15 12 1 6
   nfa ^: _1 j
i sing of olaf
   encode=. (?~# a)& i.
   encode j
22 14 21 12 23 22 20 1 22 20 17 8 1
   encode"nfa text
uvnulwvtavtqha
   test=. {:"(encode & nfa)
   test text
i sing of olaf
   enc=. '(#a)¦3+y.' :: ''
   dec=. '(#a)¦3-~y.' :: ''
   pair=. enc&:dec
   {:"(pair&nfa) text
i sing of olaf
```

UNDER. When used with two verbs *f* and *g* (as in *f"g*), the conjunction " produces a verb that applies *f under g*; that is, the monad *g* is first applied to the argument(s), *f* is applied to the result(s), and the inverse of *g* is then applied to "undo the work of *g*", or to "map the results back to the original domain".

The inverse of *g* is the function produced by *g*^:_1. Such an inverse exists for about a dozen primitive monads (such as ¦. ^ ^. *o.* < >), for a larger number of derived functions (such as *2&o. n&¦. n&* +&n +..*&M*), and for compositions of invertible functions, as illustrated by *encode&nfa* (Number From Alphabetic).

The *obverse* conjunction *&:* can also be used to specify a formal inverse, as was done for the verb *pair*.

RELATION TO OTHER APL DIALECTS

Spelling. Although orthography and spelling are superficial aspects of language, changes in them are jarring, and should be justified by significant advantages. Restriction to the ASCII alphabet simplifies keying (by avoiding a third case as well as avoiding backspacing to produce composite characters), and removes serious problems in display devices, printers, and publication.

Moreover, the spelling scheme adopted in J retains the major advantages of the special APL alphabet: it obviates reserved words and does not require spacing between words (as required in *a mod b*, contrasted with *a|b*). Finally, it provides mnemonic clues at least as effective and international as those provided by the special APL alphabet.

Although mnemonic schemes should perhaps not be studied as such, their utilization will grow more quickly for a user who is at least cognizant of their existence and their general character. Such knowledge can be gleaned from a quick review of the Language Summary of Appendix A. For example, `+.` and `*.` for *or* and *and* not only use the analogies with *plus* and *times* exploited by Boole, but the *multiply* symbol * also suggests the least common *multiple* also represented by `*.`. Finally, `<.` and `>.` for min and max suggest analogies with the relations < and >; the ^ used for power has some pedigree in mathematics (having been used by De Morgan), and leads to `^.` for the inverse (log) and to `^:` for the *conjunction* power; the / in the symbol % suggests division and leads to `%.` for matrix divide, and to *x*`%:`*y* for root (because % occurs in the equivalent expression *y*^%*x*).

Rank, Cells, and Items. The last *k* axes of an array *y* define a *cell* of rank *k*, and the rank conjunction (as in *v*`"`*k*) produces a derived verb that applies *v* to each *k*-cell of *y*. A major cell of *y* (of rank one less than the rank of *y*) is called an *item* of *y*; verbs such as catenation (,) and insertion (+/) apply between items. Consequently, the rank conjunction can provide their application to lesser cells, obviating variants such as *comma-bar* and *slash-bar*.

The utility of the rank conjunction is further enhanced by permissive frame-building, as illustrated by *i*.`"0` *i*. 5 and `>'a';'ab';'def'`.

Scans. The utility of scan (*v*\) is enhanced by applying the *monad v* to the segments rather than applying the derived monad *v*/ to them. As a consequence, the partial sums +/\ (previously denoted by +\) is read not as *plus scan*, but more properly as *sum scan*, since the phrase +/ denotes *sum*. Moreover, the *suffix scan* `\.` provides useful behaviour not provided by the prefix scan, even in conjunction with commutation (~) and reversal (`|.`).

The dyadic cases of *v*\ and *v*\. apply *v* to infixes and outfixes, determined by a window or blind of width specified by the left argument. The infixes are useful for running sums, maxima, etc., and the outfixes are useful in cases such as `-1*/\. x-r` for the derivatives of a polynomial with respect to each of its roots.

Functional Programming. Although the operators in APL provided some capacity for functional programming from the outset, the use of a functional programming style was severely hampered in many systems by the inability to:

1) Apply operators to all functions, including derived and user-defined.

2) Use the copula to assign names to entities other than nouns, as in `sum=. +/` and `inv=. 1::'x.^: _1'`

3). Indicate self-reference in recursive definition without assigning a name.

4) Use a conjunction to define verbs, adverbs, and conjunctions within a sentence.

In J these limitations have been removed.

Further Reading. The foregoing examples omit many interesting facilities of J, such as the treatment of complex numbers and the *execute* (which has rank 1 and therefore executes each row of a table argument, and whose dyadic case provides control for error handling).

For further exploration, the Language summary may be used as a guide, and translation from earlier dialects may be practiced using so-called "idiom lists", and examples from References 4-5. Finally, the appendix of [1] provides examples of the use of J in a variety of topics, but should be approached with a warning that some definitions have been changed.

REFERENCES

1. Hui, et al, APL \?, APL90 Conference Proceedings, ACM APL Quote-Quad.

2. Versions made available at the APL90 Software Exchange included IBM PC and compatibles, and Apple Mackintosh. Inquiries concerning manuals and further ports should be addressed to ISI (listed in the Vector Product Guide).

3. C.D. Olds, Continued Fractions, The Mathematical Association of America New Mathematical Library, 1963.

4. Iverson, K.E., Notation as a Tool of Thought, CACM 1980.

5. Thompson, N., APL Programs for the Mathematics Classroom, Springer-Verlag 1989.

Appendix A: Language Summary

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