A.K. Howell

APL\360

Reference Manual

Sandra Pakin

APL\360

Reference Manual

Sandra Pakin

SRA

Science Research Associates, Inc.
259 East Erie Street, Chicago, Illinois 60611

A Subsidiary of IBM

TABLE OF CONTENTS

TABLES

FIGURES

APPENDICES

INDEX

FOREWORD

APL was first defined by K. E. Iverson in A Programming Language (Wiley, 1962) and has since been further developed in collaboration with A. D. Falkoff and L. M. Breed.

This manual presents a comprehensive discussion of APL\360 as of November 1969. It is intended to provide APL users with a complete reference document. Teachers should find this manual a useful supplement to their APL texts and courses.

This manual is organized for reference rather than for structured or systematic learning. The table of contents provides a guide to the organization; the text is cross-referenced; many examples are included; and a detailed index is provided. Moreover, APL symbols and their meanings, as well as the other symbols used in this manual are catalogued in the appendices.

Peter Calingaert, now with the University of North Carolina, was a guiding force throughout the creation of this manual. I am indebted to Kenneth Iverson, Adin Falkoff, and Richard Lathwell, all of the IBM Watson Research Center, for their close critical review of an earlier version of this manual and to Gene McDonnell and Al Rose for their review of the penultimate draft of this manual. My colleagues at the Computer-Related Instructional Systems Center were always helpful--contributing ideas, asking questions, listening, and offering encouragement. In particular, I would like to thank Raymond Polivka, Harold Driscoll, Scott Krueger, and Tom McMurchie for their many useful suggestions; also Stephen Soule for the functions *SIMULATION*, *COMPRESSION*, *FINT*, and *PR* and Jules Kaplan for the functions *CODE* and *WHEREIN*.

This manual was typed, corrected and formatted using a set of text editing APL functions defined by M. M. Zryl and A. P. Mullery of the IBM Watson Research Center.

Sandra Pakin

DATA

Number Data

The 13 characters 0 1 2 3 4 5 6 7 8 9 . ¯ *E* are used in the representation of numbers.

The decimal digits 0 through 9 and the decimal point are used in the usual way. The character ¯, called the negative sign, is used to denote negative numbers. It appears as the leftmost character in the representation of any number whose value is less than zero:

```
      0-4
¯4

      ¯3-¯2
¯1
```

The negative sign, ¯, is distinct from -, the symbol used to denote subtraction. It is not a function and can be used only as part of a numeric constant. It can never be applied to an identifier:

```
      ¯B
SYNTAX ERROR
      ¯B
      ^
```

Exponential Representation. The numbers 26000, .000347, and ¯2632.15 can be written in exponential form as 26*E*3, 3.47*E*¯4, and ¯2.63215*E*3. (*E* can be read "times ten to the.") Exponential representation is written as a number immediately followed by *E* immediately followed by an integer representing an appropriate power of 10.

The examples on the next page show several numbers written in *E* notation. Observe that the same number can be represented in many ways.

Number	Exponential Representation
299654	2.99654*E*5 299.654*E*3
567.893	5.67893*E*2 567893*E*¯3
.0000056	5.6*E*¯6 56*E*¯7
¯453212.678	¯4.53212678*E*5 ¯453212678*E*¯3

No spaces may separate *E* from the numbers on either side of it. A number may be entered in either exponential or ordinary form. Which the computer will display depends on the number (see Output, page 119).

Constant Vector. A numeric constant vector is a vector of numbers each of which is separated from adjacent components by one or more spaces--for example, the expression 2.4 ¯23 567 3.5*E*4 is a constant vector with four components: two and four-tenths, negative twenty-three, five hundred sixty-seven, and thirty-five thousand.

A constant vector is treated as a single entity, just as ¯6 or 3.5 or 1.1*E*15 or 72 is a single entity. Consequently, when a constant vector occurs in an expression, it is not necessary to enclose it in parentheses:

```
      5 6 7+8
13  14  15

      3 2 1×6 5 4
18  10  4
```

A single number followed by a space does not indicate that the expression is a one-component vector:

```
      A←2
      ⍴A
BL
```

Note: Since the display of the empty vector is movement, not printing, the letters BL (for Blank Line) have been used in this manual to show that the computer response to a command is a blank line (see also Output, page 122).

Character Data

The printing keyboard symbols (including valid overstruck symbols), the space, and the carrier return can be used singly or in combination as character data. Character data is represented by enclosing a sequence of characters in single quotes. These quotes indicate that the characters keyed in are not to be regarded as symbols for numbers, variables or functions, but rather represent only themselves. The enclosing quotes are not printed in the display of a character sequence:

```
      'ABC∇123EFGΦ'
ABC∇123EFGΦ

      'PROBLEM 1:   1+2×3*4:5'
PROBLEM 1:   1+2×3*4:5

      M←'CONTINUE WHEN READY'
      M
CONTINUE WHEN READY
```

A quote, if it is to be considered character data, must be represented as a double quote:

```
      'DON''T STOP'
DON'T STOP
```

One character enclosed by quotes is a scalar. All other sequences are vectors--and behave as vectors:

```
      W←'S'          |      D←''          |      F←'A↑C'
      ρW             |      ρD            |      ρF
BL                   |  0                 |  3
```

```
      S←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      S[6 1 14 20 1 19 20 9 3]
FANTASTIC
```

Intermixing Data

Data can be intermixed for output (and only for output) by separating expressions with semicolons. Extra spaces between expressions are not inserted by the computer. The primary use for this form is to intermix numeric and character expressions:

```
      'AREA IS ';18×37;'; PERIMETER IS ';2×18+37;'.'
AREA IS 666; PERIMETER IS 110.

      'THE VALUE OF 4×15 IS ';4×15
THE VALUE OF 4×15 IS 60
```

If a matrix is one of the expressions, output will begin on a new line:

```
      'THE ANSWER IS'; 1 0/[1] 2 3 ρ'YESNO '
THE ANSWER IS
YES
```

Arrays

It is often convenient to treat many elements of data simultaneously as a single entity. This is done in APL by using arrays. Vectors and matrices, for example, are arrays. To reference an element or component of an array, one uses one or more numbers to indicate its position in the array. These position numbers will be called indices.

An array whose elements are selected by one index is called a vector. It has one coordinate and can be thought of as a collection of elements arranged in a line. A vector is normally displayed as a horizontal line (see Output, page 120):

```
      ⎕←V←6.8  12  5  ¯4  7.5
6.8  12  5  ¯4  7.5

      ⎕←V←'ABCDEFG'
ABCDEFG
```

An array whose elements are selected by two indices is called a matrix. It has two coordinates and can be thought of as a collection of elements arranged in a rectangle. A matrix is normally displayed in a rectangular pattern (see Output, page 121):

```
      ⎕←M←2 3⍴4 6 12 4 2 8

4  6  12
4  2   8

      ⎕←L←2 4⍴'ACFGIPNT'

ACFG
IPNT
```

A rank-N array (there is no special name if N exceeds 2) is one whose elements are selected by N indices. It has N coordinates and can be thought of as a collection of elements arranged along N mutually perpendicular directions (see Output, page 121, for display).

Although most arrays have more than one element, arrays of one element and of no elements also exist. An array of no elements will be called an empty array (see also Output, page 122).

In contrast to a one-component array, a scalar is a single number or single letter that cannot be indexed. It has no coordinates and can be thought of as a point.

The number of indices required to specify positions in array A is given by $\rho\rho A$ and is known as the rank of A. The vector ρA is called the dimension of A. Its Ith component is the number of possible indices for the Ith coordinate of A. The total number of elements of A is the product $\times/\rho A$. The dimensions and ranks of scalars, vectors, matrices, and rank-3 arrays are shown in Fig. 1; extensions to higher-rank arrays follow a similar pattern:

Array Type

	Scalar	Vector	Matrix	3-Array
Dimension (ρA)		N	M,N	L,M,N
Rank ($\rho\rho A$)	0	1	2	3

Fig. 1. Dimensions and Ranks of Arrays

Indices. The indices of an array are the set of integers starting with either 0 or 1, depending on the index origin (see Origin Command, page 141). The indices of the elements of a vector V are the integers $\iota\rho V$. Suppose V is a vector whose elements are 6 3.1 ¯13 5. In 1 origin, each element of V is identified by its index as follows: The element 6 is $V[1]$; the element 3.1 is $V[2]$; the element ¯13 is $V[3]$; and the element 5 is $V[4]$, or $V[\rho V]$.

The index for an element of a matrix consists of two integers--the first (left) selecting the element from a row; the second (right) selecting it from a column. Look at the matrix *D*, for example:

```
 7  4  6 ¯5
¯2  9  3  2
```

The element 7 is *D*[1;1], that is, it is in the first row, first column; the element 4 is *D*[1;2], that is, it is in the first row, second column; the element 9 is *D*[2;2], that is, it is in the second row, second column; and so forth.

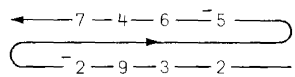
Similarly, the index for an element in a rank-N array consists of N integers, one for each coordinate of the array.

The indices of an array are ordered with the rightmost position varying fastest, then the next rightmost, and so forth. The sequence of indices of the matrix *D* is

Element	Sequence of Indices
7	*D*[1;1]
4	*D*[1;2]
6	*D*[1;3]
¯5	*D*[1;4]
¯2	*D*[2;1]
9	*D*[2;2]
3	*D*[2;3]
2	*D*[2;4]

Notice that the column index increases while the row index remains stationary. Once the column index has reached the maximum for matrix *D*, a four-column matrix, the row index increases by one and the column index starts over once again. This sequencing of the indices of an array is similar to the sequencing of words in an alphabetic listing or to the incrementing of an odometer.

Elements are restructured from an array following the index sequence. For matrix *D*, this means that the element *D*[1;1], or 7, is taken first; *D*[1;2], or 4 next; *D*[1;3], or 6 next; and so forth:



If *D* is restructured (see Restructuring, page 41 and Indexing, page 60) as a rank-3 array *T*, *T* looks like this:

```
 7  4
 6 ¯5

¯2  9
 3  2
```

The sequence of indices of *T* is as follows:

Element	Sequence of Indices
7	*T*[1;1;1]
4	*T*[1;1;2]
6	*T*[1;2;1]
¯5	*T*[1;2;2]
¯2	*T*[2;1;1]
9	*T*[2;1;2]
3	*T*[2;2;1]
2	*T*[2;2;2]

Identifiers

An identifier is used for variables, function names, group names, workspace names, and labels. It is a single letter (*A* to *Z* or *Δ* to *Δ*), Δ, or Δ, or a combination of letters, numbers, Δ, and Δ. All combinations are permitted except those beginning with a number or beginning with *SΔ* or *TΔ*. No spaces are permitted in identifiers.

UNIT *COS* *T* *Q* *ABC* *X1*

Variables, function names, groups, and labels may be of any length up to 77 characters. Workspace names are limited to 11 characters. Longer names may be used, but the additional characters are ignored.

PRIMITIVE FUNCTIONS

Primitive functions are functions that are part of the APL interpreter. A primitive function is denoted by a symbol composed of a nonalphabetic, nonnumeric character or by a combination of such characters. The five symbols ? ∨ ⌽ * ! are examples.

A primitive function may be either monadic, a function of one argument, or dyadic, a function of two arguments. The syntax of primitive functions is illustrated below.

Monadic	Dyadic
R←m *B*	*R*←*A* d *B*

m is any monadic function symbol.
d is any dyadic function symbol.

In the illustration above and in subsequent displays of function syntax, the letter *A* will represent a left argument--that is, an expression to the left of the function symbol. The letter *B* will represent a right argument--that is, an expression to the right of the function symbol. And the letter *R* will represent the result (or resultant).

There are three types of primitive functions--scalar functions, composite functions, and mixed functions.

Scalar Functions

A scalar function is a function defined on scalar arguments, yielding a scalar as a result.

Extensions to Arrays. Monadic scalar functions extend to arrays component by component. The dimensions and rank of the result is the same as that of the argument. Dyadic scalar functions extend to arrays in composite functions (see Table III) and component by component. For component by component extension of the dyadic scalar functions, arguments are conformable under any of the following conditions:

For dyadic scalar functions *A* d *B*:

1. *A* and *B* are scalar.
   The result is scalar.

```
      □←T←4-8
¯4
      ρT
BL
```

2. *A* and *B* are of the same dimension(s) and rank.
   The dimension(s) and rank of the result are the same as those of either argument.

```
      □←T←3 4 7+9 6 10
12 10 17
      ρT
3
```

3. *A* (or *B*) is a one-component array.
   The dimension(s) and rank of the result are the dimension(s) and rank of *B* (or *A*).

```
      □←T←4 8 6×1 1ρ4
16 32 24
      ρT
3
      □←S←3+0 3ρ4
BL
      ρS
0 3
```

4. *A* is a one-component array, and *B* is a one-component array.

The dimension(s) and rank of the result are the dimension(s) and rank of the argument with the greater rank.

```
      □←T←(1 1 1⍴6)+1 1⍴8

   0.75
      ⍴T
1  1  1
```

Definitions. The primitive scalar functions are defined in Table I, Monadic Scalar Functions, and Table II, Dyadic Scalar Functions. The argument(s) may be scalar, vector, matrix, or rank-N arrays, subject to the conformability requirements stated above. All functions are defined for numeric arguments. Functions that are also defined for character arguments will be so indicated by an (L) placed in the right margin beside the function syntax and by showing an example that has character arguments.

You will find the symbol ⊢ used in the definitions of all the primitive functions. This symbol is not an APL function. It is used as a metasymbol to assert that the value of the expression to its right, whether scalar, vector, matrix, or other, consists entirely of ones. For example, ⊢ *R* = *B*⌈-*B* asserts that *R* has the value of the expression *B*⌈-*B*.

Table 1--Scalar Monadic Functions

DEFINITION	EXAMPLES
IDENTITY *R←+B* ⊢ *R* = 0+*B*	+7 7 +¯4 5.3 12 ¯4 5.6 12
SIGNUM *R←×B* ⊢ *R* = (0<*B*)-0>*B* *R* is ¯1, 0, or 1 depending on whether *B* is negative, zero, or positive.	×7 1 ×¯4 5.6 0 ¯1 1 0
NEGATION *R←-B* ⊢ *R* = 0-*B*	-5 ¯5 -3 ¯4 7.6 ¯3 4 ¯7.6
RECIPROCAL *R←÷B* ⊢ *R* = 1÷*B* *B* must not be zero.	÷5 0.2 ÷4 ¯2 .11111111111 0.25 ¯0.5 9
EXPONENTIAL *R←*B* ⊢ *R* = e*B* e is the base of the natural logarithm. It is represented as approximately 2.7182818284590451	*1 2.718281828 *¯3 .5 0.04978706837 1.648721271

DEFINITION	EXAMPLES
NATURAL LOGARITHM $R\leftarrow\circledast B$ $\vdash B = \star R$, that is, $\vdash R =$ ln B B must be greater than zero.	⊛2.7182818284 1 ⊛.04978706837 1.648721271 ¯3 0.5000000002
FLOOR $R\leftarrow\lfloor B$ R is the algebraically greatest integer less than or equal to B. $\vdash R= B-1\mid B$ (See Fuzz, page 120.)	⌊6.3 6 ⌊6.7 3.1 ¯2.3 ¯7.8 6 3 ¯3 ¯8 ⌊5 ¯5 5 ¯5
CEILING $R\leftarrow\lceil B$ R is the algebraically least integer greater than or equal to B. $\vdash R = B+1\mid -B$ (See Fuzz, page 120.)	⌈6.3 7 ⌈6.7 3.1 ¯2.3 ¯7.8 7 4 ¯2 ¯7 ⌈5 ¯5 5 ¯5
ABSOLUTE VALUE $R\leftarrow\mid B$ $\vdash R = B\lceil -B$	⌈¯3.11 3.11 ⌈¯3 ¯4.5 6 3.2 3 4.5 6 3.2

DEFINITION	EXAMPLES
MONADIC RANDOM (ROLL) *R←?B* *R* is an integer pseudo-randomly selected from the integers ⍳*B*. Each number in the population has an equal chance of being selected. (See Seed, page 116.)	?8 5 ?8 8 8 8 8 8 8 8 7 4 5 2 8 1 6 4 ?6 12 25 87 8 6 5 13 73 3
NOT *R←~B* ⊢ *R* = 1≠*B* *B* must be equal to 1 or 0.	~1 0 ~1 0 1 0 1 0 1 0 1 0
GENERALIZED FACTORIAL *R←!B* When *B* is a non-negative integer, the result is *B* factorial, that is, ×/⍳*B*. For other arguments, ⊢ *R* = Γ*B*+1. *B* may not be a negative integer. Γ is a symbol for the gamma function.	!6 720 !.5 ¯.5 0 0.8862269254 1.772453851 1
PI TIMES *R←○B* ⊢ *R* = π×*B* π is represented as approximately 3.141592653589793	○1 3.141592654 ○0.25 .5 0.7853981634 1.570769327

DEFINITION	EXAMPLES
ADDITION $R\leftarrow A+B$ ⊢ $R = A+B$	2 3 4+4 5 6 6 8 10 2.3 ¯6 .5+.3 2.6 ¯5.7 0.8
SUBTRACTION $R\leftarrow A-B$ ⊢ $R = A-B$	4 6 2-3 7 ¯3 1 ¯1 5 6-4.2 ¯2 7 1.8 8 ¯1
MULTIPLICATION $R\leftarrow A\times B$ ⊢ $R = A\times B$	4 6 2×1 6 7 4 36 14 5×6 4 3.1 30 20 15.5
DIVISION $R\leftarrow A\div B$ ⊢ $R = A\div B$ If A is nonzero, B must be nonzero. If A is zero and B is zero, the result has the value 1. (lim x÷x = 1) x→0	6 20 4÷3 6 5 2 3.333333333 0.8 0÷6 0 0÷0 1
MINIMUM $R\leftarrow A\lfloor B$ ⊢ $R = A$ if A is less than or equal to B. ⊢ $R = B$ if A is greater than B.	6⌊78 6 4⌊3 4 ¯7 8 3.1 3 4 ¯7 4 3.1 2 6.5 ¯5⌊4 1 ¯9 2 1 ¯9

DEFINITION	EXAMPLES

MAXIMUM $R\leftarrow A\lceil B$
 	
⊢ $R = A$ if A is greater than B.
 	
⊢ $R = B$ if A is less than or equal to B.
	`      6⌈78`
	`78`
	
	`      4⌈3 5 ¯7 8`
	`4  5  4  8`
	
	`      2 6.5 ¯5⌈4 1 ¯9`
	`4  6.5  ¯5`

RESIDUE $R\leftarrow A\mid B$
 	
The result is the least nonnegative R such that, for some integer Q, ⊢ $B = R+A\times Q$.
 	
If A is zero, B must be nonnegative and ⊢ $R = B$.
	`      3|5`
	`2    ⊢ 5 = 2+3×1`
	
	`      3 ¯3 ¯3|¯5 4 ¯4`
	`1  1  2`
	
	`      2.5|7.24 6`
	`2.24  1`
	
	`      1|3.25 3 ¯6`
	`0.25  0  0`

CIRCULAR $R\leftarrow A\circ B$
 	
A designates the function of B as follows:
 	
`A  R  Domain`
`¯7  arctanh  1>|B`
`¯6  arccosh  B≥1`
`¯5  arcsinh`
`¯4  (¯1+B*2)*.5  1≤|B`
`¯3  arctan`
`¯2  arccos  1≥|B`
`¯1  arcsin  1≥|B`
`0  (1-B*2)*.5  B≤1`
`1  sine`
`2  cosine`
`3  tangent`
`4  (1+B*2)*.5`
`5  sinh`
`6  cosh`
`7  tanh`
 	
For the trigonometric functions, B is expressed in radians.
	`      1 2 300.5 2 .25`
	`1  1  1`
	
	`      4 0 ¯405 .25 1`
	`5.099019514  0.9682458366  0`
	
	`      5 6 700`
	`0  1  0`
	
	`      ¯1 ¯2 ¯301`
	`1.570796327  0  0.7853981634`
	
	`      ¯5 ¯6 ¯700 1 0`
	`0  0  0`

DEFINITION	EXAMPLES

GENERALIZED COMBINATION *R←A!B*
 	
The number of combinations of *B* things taken *A* at a time.
 	
⊢ *R* = (!*B*)÷(!*A*)×!*B*-*A*
 	
A!*B* is related to the beta function as follows:
 	
⊢ β (*A*,*B*) = ÷*B*×(*A*-1)!*A*+*B*-1
	`      4!8`
	`70`
	
	`      2.5 2.4!4 6.7`
	`5.432488724  24.39985591`
	
	`      0 1 2 3!3`
	`1  3  3  1`
	
	`      ¯2.1 ¯.6!2 6`
	`0.007370511731  0.1426763772`

EXPONENTIATION *R←A***B*
 	
⊢ *R* = A^B
 	
⊢ (*A**0) = 1
 	
If *A* is positive, *B* can be any value.
 	
If *A* is zero, *B* must be nonnegative.
 	
If *A* is negative, *B* must be an integer or any expression whose value is *P*÷*Q* where *P* is an integer and *Q* is an odd integer.
 	
If *A* and *B* are 0, the result is 1.
(lim x*x = 1)
x→0
	`      5 3 4*0 4 ¯2`
	`1  81  0.0625`
	
	`      0*6`
	`0`
	
	`      ¯32 ¯125*.6 .2`
	`¯8  ¯2.626527804`
	
	`      ¯243*¯.2`
	`¯0.3333333333`
	
	`      0*0`
	`1`

LOGARITHM *R←A*⊛*B*
 	
⊢ *R* = $\mathrm{Log}_A B$
 	
⊢ *R* = (⊛*B*)÷⊛*A*
 	
A and *B* must be greater than zero, and *A* cannot be equal to 1 unless *B* is also equal to 1.
	`      5 3 4⊛1 81 .0625`
	`0  4  ¯2`
	
	`      10⊛1 10 25 100`
	`0  1  1.397940009  2`

DEFINITION	EXAMPLES
LESS THAN $R\leftarrow A<B$ ⍝ R = 1 if $(A-B)$ is less than or equal to -fuzz×|B. ⍝ R = 0 otherwise.	1 4 ¯4<3 2 ¯1 1 0 1
LESS THAN OR EQUAL $R\leftarrow A\leq B$ ⍝ R = 1 if $(A-B)$ is less than or equal to fuzz×|B. ⍝ R = 0 otherwise.	1 4 ¯2≤3 4 ¯5 1 1 0
EQUAL $R\leftarrow A=B$ (L) If A and B are numbers; ⍝ R = 1 if (|$A-B$) is less than or equal to fuzz×|B. ⍝ R = 0 otherwise. If A or B is a character; R is 1 where the relationship holds, 0 where it does not.	3 4 ¯3=3 ¯4 ¯3 1 0 1 'ABC'='ADC' 1 0 1 'ABC'=145 0 0 0
GREATER THAN OR EQUAL $R\leftarrow A\geq B$ ⍝ R = 1 if $(A-B)$ is greater than -fuzz×|B. ⍝ R = 0 otherwise.	5 ¯3 4≥5 3 ¯4 1 0 1
GREATER THAN $R\leftarrow A>B$ ⍝ R = 1 if (|$A-B$) is greater than fuzz×|B. ⍝ R = 0 otherwise.	4 ¯2 3>3 ¯2 6 1 0 0

DEFINITION	EXAMPLES

NOT EQUAL $R\leftarrow A\neq B$ (L)
 	
If A and B are numbers;
⊢ R = 1 if ($\|A-B$) is greater than fuzz×$\|B$.
⊢ R = 0 otherwise.
 	
If A or B is a character;
R is 1 where the relationship holds, 0 where it does not.
	`      4 ¯2 3≠3 ¯2 6`
	`1 0 1`
	
	`      'ABC'≠'ADC'`
	`0 1 0`
	
	`      'ABC'≠145`
	`1 1 1`

AND $R\leftarrow A\wedge B$ ⊢ R = 1 if ⊢ A = 1 and ⊢ B = 1. ⊢ R = 0 otherwise. A and B must be 0 or 1.	The table below shows the result for each combination of arguments: ∧ | 0 1 0 | 0 0 1 | 0 1
OR $R\leftarrow A\vee B$ ⊢ R = 0 if ⊢ A = 0 and ⊢ B =0. ⊢ R = 1 otherwise. A and B must be 0 or 1.	The table below shows the result for each combination of arguments: ∨ | 0 1 0 | 0 1 1 | 1 1
NAND $R\leftarrow A\uparrow B$ ⊢ R = $\sim A\wedge B$ A and B must be 0 or 1.	The table below shows the result for each combination of arguments: ↑ | 0 1 0 | 1 1 1 | 1 0
NOR $R\leftarrow A\downarrow B$ ⊢ R = $\sim A\vee B$ A and B must be 0 or 1.	The table below shows the result for each combination of arguments: ↓ | 0 1 0 | 1 0 1 | 0 0

Composite Functions

Reduction, inner product, and outer product--the three composite functions-- are extensions to arrays of the dyadic scalar functions.

Definitions. For the composite functions and for the mixed functions defined in Tables IV and V, the greatest rank permitted for each argument is given as well as expressions for the dimensions $\circ R$ and rank $\rho\rho R$ of the result. Examples and definitions show the behavior of arrays up to rank 3. The extension to arrays of rank greater than 3 is similar.

Example: In the column headed SYNTAX and RANK in Table V, the following appears in the definition of Catenation (page 42):

$R \leftarrow A , B$

V V

$\vdash (\rho R) = (\rho , A) + \rho , B$

$\vdash (\rho\rho R) = 1$

Line 1 is the syntax of the function.
Line 2 indicates that the maximum arrays permitted as arguments are vectors.
Line 3 is the dimension of the result.
Line 4 is the rank of the result.

The abbreviations used to describe the ranks are

S rank 0: scalar or any one-component array.
V rank 1: vector (Scalar is permitted.)
M rank 2: matrix (Vector or scalar is permitted.)
H rank-N array (Matrix, vector, or scalar is permitted.)

In some of the function definitions a bracketed value follows to the immediate right of the function symbol:

$R\leftarrow m[K]B$ | $R\leftarrow A\,d[K]B$

This bracketed value stands for a coordinate of the argument array and is used to specify the coordinate along which the function is to operate. K must be an integer in the set $\iota\rho\rho B$. If $[K]$ is elided, the last coordinate is assumed.

Definitions generally are written in terms of origin 1. If you are using origin 0, the coordinates of the array are numbered from 0 instead of from 1 (see Origin Command, page 141).

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

REDUCTION

The result is an array whose dimension vector ρR is equal to that of B with the Kth component suppressed. If the argument has nonzero rank, the rank of the result will be one less than the rank of the argument.

If the Kth component of the dimension vector is the last component, it may be elided. Thus the expression d/B is equivalent to d/[$\rho\rho B$]B. (The character d stands for any dyadic scalar function.)

$R \leftarrow$ d/[K]B

H

⊢ (ρR) = ($K \neq \iota \rho\rho B$)/ρB

⊢ ($\rho\rho R$) = $0\lceil{}^{-}1+\rho\rho B$

Vector Argument

For ρB nonzero and ρB not 1, the value of the result is that of the expression which results from placing the dyadic scalar function d between each pair of adjacent components of the vector B.

⊢ R = B[1] d B[2] d ... d B[ρB]

```
      □←R←+/3 4 6 9 4
26      ⊢ (3+4+6+9+4) = 26

      ρR
BL


      -/4 3 ¯4 6 ¯8
¯17     ⊢ (4-3-¯4-6-¯8) = ¯17


      ⌈/7 32 ¯45 63 10
63    ⊢ (7⌈32⌈¯45⌈63⌈10) = 63

      ∧/1 0 0 1 1
0

      ∧/1 1 1 1 1
1

      </¯8 9 3 2
1
```

Note: Minus reduction -/B yields an alternating sum; divide reduction ÷/B, an alternating product.

Rank-N Argument

Reduction of the Kth coordinate.

```
      □←M←3 4⍴⍳12
 1  2  3  4
 5  6  7  8
 9 10 11 12

      +/M
10 26 42

      +/[1]M
15 18 21 24

      □←H←3 4 5⍴1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

      +/[1]H
3 3 3 3 3
3 3 3 3 3
3 3 3 3 3
3 3 3 3 3

      +/[2]H
4 4 4 4 4
4 4 4 4 4
4 4 4 4 4

      +/H
5 5 5 5
5 5 5 5
5 5 5 5
```

Scalar and One-Component Array Argument
The result has the same value as B.

```
      □←R←+/7
7

      ⍴R
BL

      ∧/8
8

      □←S←+/1 1⍴7
7

      ⍴S
1

      !/12
12
```

No-Component Array Argument. If the argument is the empty vector (or an empty array), the result has the value of the identity element of the function--if one exists. These identity elements are listed below:

FUNCTION	IDENTITY ELEMENT	NOTES
×	1	
+	0	
÷	1	Right identity only
-	0	Right identity only
*	1	Right identity only
|	0	Left identity only
⊛		No identity element
○		No identity element
∨	0	
∧	1	
⍲		No identity element
⍱		No identity element
!	1	Left identity only
⌈	¯7.237005577E75	Minimum number representable
⌊	7.237005577E75	Maximum number representable

The following apply to arguments in the domain 0 1 only

>	0	Right identity only
≥	1	Right identity only
<	0	Left identity only
≤	1	Left identity only
=	1	
≠	0	

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

INNER PRODUCT

The dimension vector of the result is the vector of all but the last component of ρA catenated with all but the first component of ρB.

The arguments are conformable if A or B is a scalar or
$\vdash (\rho A)[\rho\rho A] = (\rho B)[1]$.

The letters d and D represent any dyadic scalar functions. The composite +.× represents the ordinary matrix product.

Vector Arguments

```
⊢ R = d/A  D B
```

```
      5+.×3 2
25

      1 2 4 8-.÷1 1 2 6
¯0.3333333333
```

Vector and Matrix

```
⊢ R[I] = d/A  D B[;I]
```

Matrix and Vector

```
⊢ R[I] = d/A[I;]  D B
```

```
      6 8 5+.×3 2ρ6 3 8 5 2 6
110  88

      (3 3ρ1 0)∨.∧0 0 1
1  0  1

      1 2 3+.!3 3ρ0 0 1 1 2 2 4 4 4
4  5  6
```

```
R←Ad.DB     (L)
  H   H
```

⊢ $(\rho R) = (^{-}1\downarrow\rho A), 1\downarrow\rho B$

⊢ $(\rho\rho R) = {}^{-}2+(\rho\rho A)+\rho\rho B$

Matrix and Matrix

```
⍺ R[I;J] =    d/A[I;] ∘.D B[;J]
```

```
      (3 2⍴⍳6)+.×2 3⍴⍳6
 9  12  15
19  26  33
29  40  51

      (2 2⍴'SEAT')∧.=2 2⍴'EAST'
0 0
0 1
```

Other Array Arguments

The examples below illustrate the inner product for other array arguments.

```
      (3 2 3⍴⍳18)+.×100 10 1
 123  456
 789 1122
1455 1788

      (2 2 2⍴⍳8)+.-2 2 2⍴⍳8
¯3 ¯5
¯7 ¯9

 1 ¯1
¯3 ¯5

 5 ¯3
 1 ¯1

 9  7
 5  3
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

OUTER PRODUCT

The dimension vector of the result is ρA catenated with ρB.

Every component of A d every component of B, where d is any scalar dyadic function.

$R \leftarrow A \circ .dB$ (L)

H H

$\vdash (\rho R) = (\rho A), \rho B$

$\vdash (\rho\rho R) = (\rho\rho A) + \rho\rho B$

Vector Arguments

$\vdash R[I;J] = A[I]$ d $B[J]$

```
      1 2 3 4∘.×1 2 3 4 5

1  2   3   4   5
2  4   6   8  10
3  6   9  12  15
4  8  12  16  20

      'ABCD'∘.≠'AECF'

0  1  1  1
1  1  1  1
1  1  0  1
1  1  1  1

      1 2 3 4∘.=1 2 3 4

1  0  0  0
0  1  0  0
0  0  1  0
0  0  0  1

      3 6∘.÷5 9

0.6  0.3333333333
1.2  0.6666666667

      5 9∘.÷3 6

1.666666667  0.8333333333
3            1.5
```

Other Array Arguments

An example is shown below.

```
      (2 2ρ6 12 45 8)∘.⌊0 67 13

0   6   6
0  12  12

0  45  13
0   8   8
```

Mixed Functions

All primitive functions that are not listed as scalar or composite functions are called mixed functions.

Definitions. Unlike the scalar functions, whose primary definitions can be stated completely in terms of scalar arguments and a scalar result, the primary definitions of mixed functions always involve vectors either as arguments or as the result. Consequently, the extension of mixed functions to arguments of higher rank is not as uniform as the extension of scalar functions to arguments of higher rank.

An explanation of the abbreviations used to denote the dimension(s) and rank of the result of each function will be found on page 28. As with the scalar functions, mixed functions that are defined for character arguments as well as for numeric arguments will be so indicated by an (L) placed in the right margin beside the function syntax and by showing an example that has character arguments.

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

INDEX GENERATOR

The result is a vector of all the possible indices in ascending order of a vector with *B* components, starting with the index origin.

If the argument is zero, the result is the empty vector.

```
      ⍳5
1  2  3  4  5
```

The expression ⍳0 is a common expression whose value is the empty vector.

Note: A system command can make the origin 0 (see page 141). With origin 0, the first component of ⍳*B* is 0 and the last component is *B*-1:

```
      )ORIGIN 0
WAS 1
      ⍳5
0  1  2  3  4
```

R←⍳*B*

S

⊢ (⍴*R*) = *B*

⊢ (⍴⍴*R*) = 1

RAVEL

The result is a vector whose components are the components of the right argument taken in index sequence (see page 14).

⊢ *R* = (×/⍴*B*)⍴*B*

```
      T←,5
      ⍴T
1

      M←2 3⍴⍳6
      M
  1  2  3
  4  5  6

      ,M
1  2  3  4  5  6
```

```
      E←2 3⍴'ABCDEF'
      E
ABC
DEF

      ,E
ABCDEF
```

R←,*B* (L)

H

⊢ (⍴*R*) = ×/⍴*B*

⊢ (⍴⍴*R*) = 1

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

DIMENSION (SIZE)

The result is a vector whose components represent the number of values each index of *B* has. The expression ρρ*B* gives the rank of *B*. In 1 origin, ρ*B* gives the highest index in each coordinate. (See also Arrays, page 13.)

```
      B←6 8 3 5          C←2 3⍴⍳6
      ⍴B                 ⍴C
4                  2 3

      ⍴⍴B                ⍴⍴C
1                  2

      D←'AEFG'           W←3 4 5⍴⍳60
      ⍴D                 ⍴W
4                  3 4 5

      T←5                T←⍳0
      ⍴T                 ⍴T
BL                 0
```

R←ρ*B* (L)

H

⊢ (ρ*R*) = ρρ*B*

⊢ (ρρ*R*) = 1

REVERSAL

The result is the reversal of the *K*th coordinate of the argument.

If *B* is a vector,
⊢ *R* = *B*[1+(ρ*B*)-⍳ρ*B*].

For all other arrays,
⊢ *R* = *B*[; ;⌽⍳(ρ*B*)[*K*]; ;].

```
      V←4 5 3 2 6        C←'LIVE'
      ⌽V                 ⌽C
6 2 3 5 4          EVIL

          M←3 4⍴⍳12

      ⌽M                 ⌽[1]M

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

R←⌽[*K*]*B* (L)

H

⊢ (ρ*R*) = ρ*B*

⊢ (ρρ*R*) = ρρ*B*

DEFINITION AND EXAMPLES | DIMENSIONS AND RANKS

MONADIC TRANSPOSITION

R←⍉*B* (L)

H

The result is an array like the argument with its last two coordinates interchanged.

For a matrix the result is a matrix whose rows are the columns of *B* and whose columns are the rows of *B*.

For ⊢ *S* = -2↓⍴⍴*B*
⊢ (⍴*R*) =
(*S*↓⍴*B*),⌽*S*↑⍴*B*

⊢ (⍴⍴*R*) = ⍴⍴*B*

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

      ⍴M
3 4
```

```
      M⍉←⍉M
      M
 1  5  9
 2  6 10
 3  7 11
 4  8 12

      ⍴M
4 3
```

```
      N←4 3⍴'LOBONEICENET'
      N
LOB
ONE
ICE
NET
```

```
      ⍉N
LOIN
ONCE
BEET
```

```
      H←2 3 4⍴⍳24
      H
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24

      ⍴H
2 3 4
```

```
      □←H⍉←⍉H
 1  5  9
 2  6 10
 3  7 11
 4  8 12

13 17 21
14 18 22
15 19 23
16 20 24

      ⍴H
2 4 3
```

DEFINITION AND EXAMPLES	DIMENSIONS AND RANKS

GRADE UP
 	
$R[I]$ is the index of the Ith smallest component of B:
 	
`      ⍋6 8 ¯8 2 1 7`
`3  5  4  1  6  2`
 	
The ranking of duplicate components of B is determined by their positions in B:
 	
`      ⍋8 8 8 8`
`1  2  3  4`
 	
`      ⍋6 3 5 3 9 3 1`
`7  2  4  6  3  1  5`
 	
Note: The result depends on the index origin (see page 141).
	$R←⍋B$
	
	V
	
	--- 	
	
	$⊢(⍴R) = ⍴B$
	
	$⊢(⍴⍴R) = ⍴⍴B$

GRADE DOWN
 	
$R[I]$ is the index of the Ith largest component of B:
 	
`      ⍒6 8 ¯8 2 1 7`
`2  6  1  4  5  3`
 	
The ranking of duplicate components of B is determined by their positions in B:
 	
`      ⍒8 8 8 8`
`1  2  3  4`
 	
`      ⍒6 3 5 3 9 3 1`
`5  1  3  2  4  6  7`
 	
Note: The result depends on the index origin (see page 141).
	$R←⍒B$
	
	V
	
	--- 	
	
	$⊢(⍴R) = ⍴B$
	
	$⊢(⍴⍴R) = ⍴⍴B$

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

RESTRUCTURING (RESHAPE)

$R \leftarrow A \rho B$ (L)

V H

The result is an array whose dimensions are A and whose elements, if any, are taken from B in the order defined by $,B$.

⍴ (⍴R) = A

⍴ (⍴⍴R) = ⍴,A

The components of A may be either positive integers or zero. If A is an empty vector, the result is a scalar. If any element of A is 0, the result is an empty array.

```
      5⍴4 2 1 7 9
4 2 1 7 9

      2 2⍴6 3 5 1
6 3
5 1

      □←M←2 4⍴7 2 ¯1 3 9 7 4 1
 7  2 ¯1  3
 9  7  4  1

      8⍴M
7 2 ¯1 3 9 7 4 1

      □←B←(⍳0)⍴5
5

      2 2 2⍴M
 7  2
¯1  3

 9  7
 4  1

      ⍴B
BL

      0⍴5
BL

      3 0⍴2
BL

      ⍴3 2⍴⍳6
3 2

      2 4⍴'STOPSIGN'
STOP
SIGN
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

If $(\times/A)>\rho B$, the sequence of components from B is repeated.

```
      4ρ10
10  10  10  10

      5ρ1 0 1
1  0  1  1  0

      12ρ'OH! '
OH! OH! OH!
```

```
      2 3ρ⍳4
1  2  3
4  1  2

      3 3ρ1 0
1  0  1
0  1  0
1  0  1
```

If $(\times/A)<\rho B$, the first $\times/A$ components needed are used.

```
      4ρ2 6 3 2 1
2  6  3  2

      3ρ'TINSEL'
TIN
```

```
      2 2ρ7 3 1 6 8 8
7  3
1  6
```

CATENATION

The result is the vector formed by appending the components of B to the components of A. Both A and B must be either numbers or characters.

$R\leftarrow A,B$ (L)

V V

⊢ $(\rho R) = (\rho,A)+\rho,B$

⊢ $(\rho\rho R) = 1$

```
      A←3
      B←4
      A,B
3  4

      E←⍳0
      ⎕←E←A,E
3

      ρE
1

      H←'QUARTER'
      G←'BACK'
      H,G
QUARTERBACK
```

```
      C←2 3 4
      D←5 1 6 3 4
      C,D
2  3  4  5  1  6  3  4

      F←3,4,,,,5,6
      ρF
4

      F
3  4  5  6

      L←' '
      H,L,G
QUARTER BACK
```

DEFINITION AND EXAMPLES | DIMENSIONS AND RANKS

ROTATION

The result is like the right argument except that the components are cyclically left-rotated $(\rho B)|A$ components. In other words, if A is positive, components are rotated to the left; if A is negative, components are rotated to the right.

$R \leftarrow A \phi [K] B$ (L)

S	H
H-1	H

$\vdash (\rho R) = \rho B$

$\vdash (\rho\rho R) = \rho\rho B$

Vector Right Argument

$\vdash R = B[1+(\rho B)|{}^{-}1+A+\iota\rho B]$

```
      3φι5          |      ¯3φι5
4  5  1  2  3      |3  4  5  1  2

      3φ'INGRAIN'   |      ¯1φ'TOPS'
RAINING             |STOP
```

Matrix and Rank-2 Right Argument

Rotation along the Kth coordinate. If $\vdash K = \rho\rho B$, it may be elided.

```
      3φ3 4ρι12     |      ¯2φ[1]3 4ρι12

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

```
      □←M←2 5ρ'GLEEAPLEAP'

GLEEA
PLEAP
```

```
      ¯2φM          |      3φM

EAGLE               |EAGLE
APPLE               |APPLE
```

```
      0 1 2 3⌽[1]3 4⍴⍳12

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

     (-1 2 3)⌽3 4⍴⍳12

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

```
    1 ¯3⌽2 4⍴⍳8
2 3 4 1
6 7 8 5
```

```
    2 3⌽2 5⍴'RPSHA'
SHARP
HARPS
```

```
    2 3 3⍴⍳18
 1  2  3
 4  5  6
 7  8  9

10 11 12
13 14 15
16 17 18
```

```
    2⌽2 3 3⍴⍳18
 3  1  2
 6  4  5
 9  7  8

12 10 11
15 13 14
18 16 17
```

```
    H←2 3 4⍴⍳24
    H
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
```

```
    N←2 4⍴3
    N⌽[2]H
 5 10  3  8
 9  2  7 12
 1  6 11  4

21 14 19 24
13 18 23 16
17 22 15 20
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

INDEX OF

$R \leftarrow A \iota B$ (L)

V H

For each element of the right argument $B[J]$, the corresponding element in the result $R[J]$ is the smallest index I such that $\vdash A[I] = B[J]$.

A cannot be scalar.

```
      4 7 3⍳7          |      4 7 4 3⍳4
2                      |1
      'ABCD'⍳'B'       |      'ABAD'⍳'A'
2                      |1
```

If no component of A has the same value as B, then $R = 1+\lceil/\iota\rho A$.

If A is the empty vector, then $R = \iota 1$.

```
      4 3 5 2⍳8        |      5 6 2⍳1
5                      |4
      'ABCD'⍳'E'       |      (⍳0)⍳7
5                      |1


      3 6 2⍳2 4        |      'ABCDEFG'⍳'SRCA'
3  4                   |8  8  3  1
      M←3 3ρ⍳9         |      L←2 3ρ'TIPTOP'
      A←6 7 9 2 1      |      L
      A⍳M              |TIP
  5  4  6              |TOP
  6  6  1              |
  2  6  3              |      'TAP'⍳L

      4 7 9⍳⍳0         |   1  4  3
BL                     |   1  4  3
```

Note: The result depends on the index origin (see Origin Command, page 141).

```
      )ORIGIN 0
WAS 1
      3 4 5⍳2 3 4
3  0  1
```

$\vdash (\rho R) = \rho B$

$\vdash (\rho\rho R) = \rho\rho B$

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

BASE VALUE (DECODE)

R←*A*⊥*B*

V V

The result *R* is the value of the right argument evaluated in a numeral system with radices *A*.

⍴ (⍴*R*) = BL

⍴ (⍴⍴*R*) = 0

If *A* is a scalar or ⍴ *A* = (⍴*B*)⍴*A*[1], then *R* is the value of a polynomial in *A* with coefficients *B*. If also ⍴ *B*<*A*, the result is the base-10 value of *B* in base *A*.

```
      8⊥2 4 6 3
1331
(⍴ 1331 = (2×8*3)+(4×8*2)+(6×8*1)+3×8*0  )
```

```
      3⊥6 4 1 3          |       5⊥3 4 2 3 2
204                      |2442

      2⊥1 0 1 1 1        |       ¯2⊥¯1 1 0 1
23                       |13

      10⊥7 6 3           |       5 5 5⊥4 2.3 7.47
763                      |118.97

      10 10 10⊥4         |       .5⊥34 12 56
444                      |70.5
```

In general, if *A* is a vector, ⍴ *R* = *B*+.×*W*, where *W* is a weighting vector of the positional value of each digit of *A*: ⍴ *W* = ⌽×\1,⌽1↓*A*. ×\ (an unimplemented function) is a product scan: for *R*←×*B*, ⍴ *R*[*I*] = ×/(,*B*)[*I*]

```
      5 60 60⊥1 2 3
3723  (time in seconds of 1 hour
      2  minutes 3 seconds)

(⍴ 3723 = 3600 60 1+.×1 2 3  )
```

```
      1 3 12⊥3 1 6
126   (measurement in inches of 3
      yards 1 foot 6 inches)
```

Note: The value of *A*[1] is immaterial in determining base value.

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

REPRESENTATION (ENCODE)

R←*A*⊤*B*

V S

The result *R* is a base *A* representation of *B*. Thus if ⊢ *R* = *A*⊤*B*, then ⊢ ((×/*A*)|*B*-*A*⊥*R*) = 0.

⊢ (ρ*R*) = ρ*A*

⊢ (ρρ*R*) = ρρ*A*

For single radix representation, the most significant digits of the result will be truncated if ⊢ (ρ*A*) < ⌊1+*A*⊛*B*.

```
      (5ρ5)⊤2442  |       (5ρ2)⊤23
3  4  2  3  2    |  1  0  1  1  1

      (4ρ3)⊤204   |       (3ρ2)⊤23
1  1  2  0       |  1  1  1

      (5ρ¯2)⊤13   |       (4ρ5)⊤118.97
1  1  1  0  1    |  0  4  3  3.97

      (4ρ10)⊤¯34  |       (7ρ2)⊤13
9  9  6  6       |  0  0  0  1  1  0  1
```

```
      1780 3 12⊤126
3  1  6  (yards, feet, inches
          in 126 inches)

      24 60 60⊤3723
1  2  3 (hours, minutes, seconds
          in 3723 seconds)
```

Note: If ⊢ *A* = 0,*A*, then *R*[1] is the part of *B* that would have been truncated had you taken *A*⊤*B*.

```
      0 1⊤13.2    |       0 10 10⊤4675
13  0.2           |  46  7  5
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

COMPRESSION

The function selects from *B* those components that correspond to unit components of *A*.

If either *A* or *B* is a one-component array or a scalar, it is extended to apply to all components of the other argument. If *A*(*B*) is a one-component array or a scalar and *B*(*A*) is empty, the result is empty.

```
⊢ (ρA) = (ρB)[K]

⊢ A ∈ 0 1
```

Vector Right Argument

```
      U←1 0 1 0
      U/5 2 3 7
5 3
      U/'ABCD'
AC
```

```
      □←R←0 0 1/6 4 12
12

      ρR
1
```

```
      B←5
      ρB
BL

      □←R←1/B
5
      ρR
1
```

Matrix and Rank-3 Right Argument

Compression along the *K*th coordinate.

If ⊢ *K* = ρρ*B*, it may be elided.

```
      0 1 0/3 3ρ⍳9
2
5
8
```

```
      0 1 0/[1]3 3ρ⍳9
4  5  6
```

R←*A*/[*K*]*B* (L)

V H

```
⊢ (ρR) =
((K-1)↑ρB),(+/A),K↓ρB

⊢ (ρρR) = ρρB
```

```
      V←1 0 1

   V/3 3⍴⍳9  │  V/[1]3 3⍴⍳9
             │
1  3         │  1  2  3
4  6         │  7  8  9
7  9         │
```

```
      ⎕←M←3 3⍴'BAAURNSKY'

BAA
URN
SKY

   V/M       │  V/[1]M
             │
BA           │  BAA
UN           │  SKY
SY           │
```

```
      ⎕←H←2 3 4⍴⍳24

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

13 14 15 16
17 18 19 20
21 22 23 24
```

```
   1 0 1/[2]H  │  1 1 0 0/H
               │
 1  2  3  4    │  1  2
 9 10 11 12    │  5  6
               │  9 10
13 14 15 16    │
21 22 23 24    │ 13 14
               │ 17 18
               │ 21 22
```

```
      0 1/[1]H

13 14 15 16
17 18 19 20
21 22 23 24
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

EXPANSION

The arguments are conformable if $\vdash (+/A) = (\rho B)[K]$.

$\vdash A \in 0\ 1$

Vector Right Argument

$\vdash R[(\sim A)/\iota\rho A] = 0$ (Or ' ' for literal data)

$\vdash R[A/\iota\rho A] = B$

```
      U←1 0 1 0 1

      U\6 3 2          U\'ABC'
6 0 3 0 2          A B C

      1 0 0\5          0 0 0 1\'S'
5 0 0                    S
```

Matrix and Rank-3 Right Argument

Expansion along the Kth coordinate.

If $\vdash K = \rho\rho B$, it may be elided.

```
      V←1 0 0 1

      V\2 2ρι4         V\[1]2 2ρι4

1 0 0 2            1 2
3 0 0 4            0 0
                   0 0
                   3 4

      V\2 2ρ'ABCD'     V\[1]2 2ρ'ABCD'

A B                AB
C D

                   CD
```

```
R←A\[K]B     (L)

  V    H
```

$\vdash (\rho R) = ((K-1)\uparrow\rho B),(\rho,A),K\downarrow\rho B$

$\vdash (\rho\rho R) = \rho\rho B$

```
      H←2 3 4⍴⍳24

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

13 14 15 16
17 18 19 20
21 22 23 24

      1 0 1\[1]H
 1  2  3  4
 5  6  7  8
 9 10 11 12

 0  0  0  0
 0  0  0  0
 0  0  0  0

13 14 15 16
17 18 19 20
21 22 23 24
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

DYADIC TRANSPOSITION

$R \leftarrow A \lozenge B$ (L)

V H

The result is an array similar to B except that the coordinates $\iota\rho\rho B$ of B are permuted according to A.

$\vdash (\rho A) = \rho\rho B$

Case 1: $\vdash A \in \iota\rho\rho B$

A is the inverse permutation vector. The $A[I]$th component of the dimension vector of the result is the Ith component of the dimension vector of B.

$\vdash (\rho R)[A] = \rho B$

The $A[I]$th coordinate of the result is the Ith coordinate of B.

```
      □←T←4 5ρι20

 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20

      2 1⌽T

 1  6 11 16
 2  7 12 17
 3  8 13 18
 4  9 14 19
 5 10 15 20

      □←U←3 4ρ'HA'

HAHA
HAHA
HAHA

      2 1⌽U

HHH
AAA
HHH
AAA
```

For a matrix this amounts to interchanging rows and columns, $\vdash (2\ 1 \lozenge B) = \lozenge B$ (see Monadic Transposition, page 39).

For
$\vdash I \in \iota\rho\rho R$
$\vdash (\rho R)[I] = \lfloor/(A=I)/\rho B$

$\vdash (\rho\rho R) = \lceil/A$

Determining the transposition for a rank-N array might be done as follows:

```
      □←S←2 2 2⍴⍳8

1 2
3 4

5 6
7 8

      R←2 3 1⍉S
```

The first coordinate of S becomes the second coordinate of R (that is, $A[1]$). The second coordinate of S becomes the third coordinate of R (that is, $A[2]$). And the third coordinate of S becomes the first coordinate of R (that is, $A[3]$). Or ⊢ $R[K;I;J] = S[I;J;K]$.

```
      R

1 3
5 7

2 4
6 8
```

It might be helpful to construct the result as follows:

1. Define T as ⍋A

   ```
   ⊢ T = 3 1 2     (For R←2 3 1⍉S)
   ```

2. The $T[I]$th coordinate of the array becomes the Ith coordinate of the result.

   In the example, the third coordinate of S becomes the first coordinate of the result; the first coordinate of S becomes the second coordinate of

the result; the second coordinate of S becomes the third coordinate of the result.

```
1  3
5  7

2  4
6  8
```

Or

2. Write down the components of B with the $T[1]$ coordinate indices varying last, the $T[2]$ coordinate indices varying next to last,...,and the $T[\rho T]$ coordinate indices varying first.

```
B[1;1;1]
B[1;2;1]
B[2;1;1]
B[2;2;1]
B[1;1;2]
B[1;2;2]
B[2;1;2]
B[2;2;2]
```

Structure the result in index sequence (see Indices, page 14), taking the components from B in the order listed above:

```
1  3
5  7

2  4
6  8
```

Another example is shown on the next page.

```
      □←F←2 3 4⍴⍳24

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

13 14 15 16
17 18 19 20
21 22 23 24

      R←2 1 3⍉F

      ⍴R
3 2 4

      R

 1  2  3  4
13 14 15 16

 5  6  7  8
17 18 19 20

 9 10 11 12
21 22 23 24
```

Case 2: ⊢ (⍳⌈/*A*)∈*A* (diagonal plane)

In a matrix the first (and only) coordinate of the result is made up of those components of *B* whose first and second coordinate indices are the same, that is, the main diagonal of the matrix.

```
      M←4 4⍴⍳16
      R←1 1⍉M
```

M[1;1], *M*[2;2], *M*[3;3], and *M*[4;4] are the components whose first and second indices are the same:

```
      R
1 6 11 16

      S←3 4⍴⍳12
      □←E←1 1⍉S
1 6 11
```

```
      ρE
3

      □←Q←3 3ρ'DOEOILELM'

DOE
OIL
ELM

      1 1⌽Q
DIM
```

In general, if ⊢ 1<+/A[I] = A and ⊢ N←(A[I]∈A)/⍳ρA, then the A[N[I]] th coordinate of the result is made up of those components of B whose Nth coordinate indices are the same. All other coordinates of the result are structured as in case 1.

```
      □←H←2 2 2ρ⍳8

  1  2
  3  4

  5  6
  7  8

      R←1 1 1⌽H
```

H[1;1;1] and H[2;2;2] are the components whose first, second, and third coordinate indices are the same.

```
      R
1  8

      R←1 2 1⌽H
```

H[1;1;1], H[1;2;1], H[2;1;2], and H[2;2;2] are the components whose first and third coordinate indices are the same. These components make up the first coordinate of R.

```
      R

  1  3
  6  8
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

TAKE

$R \leftarrow A \uparrow B$ (L)

V H

A is an integer or a vector of integers.

⊢ (ρ*A*) = ρρ*B*

⊢ (ρ*R*) = |*A*

⊢ (ρρ*R*) = ρρ*B*

Vector Right Argument

For ⊢ *A*>0:
If ⊢ *A*≤ρ*B*, *R* is a vector of the first *A* components of *B*.
If ⊢ *A*>ρ*B*, *R* is the vector *B* catenated with *A*-ρ*B* zeros (or spaces).

For ⊢*A*≤0:
If ⊢ (|*A*)≤ρ*B*, *R* is a vector of the last |*A* components of *B*.
If ⊢ (|*A*)>ρ*B*, *R* is the vector *B* preceded by (|*A*)-ρ*B* zeros (or spaces).

```
      V←6 8 9 1 2 3 7 5

      3↑V
6 8 9
      ¯3↑V
3 7 5
      ¯2↑'FLOOR'
OR
      4↑'MEETING'
MEET
      0↑V
BL
      5↑7 9 3
7 9 3 0 0
      ¯5↑7 9 3
0 0 7 9 3
      ¯8↑'STIR'
  STIR
```

Other Right Arguments

Similar to vector argument with each component of *A* corresponding to a coordinate of *B*.

```
      M←3 4ρ⍳12

      2 2↑M
1 2
5 6
      2 ¯3↑M
2 3 4
6 7 8

      ¯4 6↑M
0  0  0  0  0  0
1  2  3  4  0  0
5  6  7  8  0  0
9 10 11 12  0  0
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

DROP

R←*A*↓*B* (L)

V H

⊢ (ρ*R*) = (ρ*B*)-|*A*

⊢ (ρρ*R*) = ρρ*B*

A must be an integer or vector of integers.

Vector Right Argument

If ⊢ *A*≥0, *R* is a vector of all but the first *A* components of *B*.

⊢ *R* = *B*[*A*+⍳(ρ*B*)-*A*]

If ⊢ *A*<0, *R* is a vector of all but the last |*A* components of *B*.

⊢ *R* = *B*[⍳*A*+ρ*B*]

```
          V←6 8 9 1 2 3 7 5

      3↓V                  ¯3↓V
1 2 3 7 5              6 8 9 1 2

      1↓'BLAST'            ¯2↓'STATELY'
LAST                   STATE

      (ρV)↓V               0↓V
BL                     6 8 9 1 2 3 7 5
```

Other Right Arguments

⊢ (ρ*A*) = ρρ*B*

If ⊢ *A*[*I*]≥0, the *I* th coordinate of the result is all but the first *A*⌈*I*] components in the *I*th coordinate of *B*.

If ⊢ *A*[*I*]<0, the *I* th coordinate of the result is all but the last |*A*[*I*] components in the *I*th coordinate of *B*.

```
          □←M←3 4ρ⍳12
 1  2  3  4
 5  6  7  8
 9 10 11 12

    2 2↓M                2 ¯3↓M

11 12                    9
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

MEMBERSHIP

```
⊢ R = ∨/A∘.=B
```

```
      3∈⍳5                    6∈⍳5
1                           0

      A←6 5 3 2               E←'ABCDEF'
      B←1 2 6 3 6             G←'FGTADE'
      A∈B                     E∈G
1 0   1 1                   1 0 0 1 1 1

      D←2 3⍴⍳6                A∈E
      D∈A                   0 0 0 0

  0 1 1
  0 1 1
```

(See also Fuzz, page 120.)

```
R←A∈B      (L)
  H H
```

```
⊢ (⍴R) = ⍴A
⊢ (⍴⍴R) = ⍴⍴A
```

DYADIC RANDOM (DEAL)

The result is a vector of A components selected pseudo-randomly without replacement (thereby preventing duplicates) from the integers $⍳⍴B$.

```
⊢ A≤B
```

```
      3?10                    5?5
6 2 5                       1 3 4 5 2
```

Note: The result depends on the index origin (see Origin Command, page 141).

```
R←A?B
  S S
```

```
⊢ (⍴R) = A
⊢ (⍴⍴R) = 1
```

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

INDEXING

Indexing is included in this table as a dyadic function. You will find as you study the definition that there are some things that make indexing unlike the other primitive functions. There are three outstanding differences: (1) the function symbol is made up of two distinct symbols which surround the right argument; (2) the generalized argument representing the indices consists of expressions separated by semicolons; and (3) indexing can appear to the left of a specification.

The right argument (within brackets) is an expression or list of expressions. The *K*th such expression is an array, each of whose elements is a permissible index for the *K*th coordinate of array *A*.

Vector Left Argument

R is the result of selecting from vector *A* those components whose indices are *B*.

```
      A←1.6 5 ¯4 2 7 3 ¯.5 0 1

      □←C←A[3]
¯4

      ρC
BL

      A[1 3 5]
1.6 ¯4 7

      A[5 8 2]
7 0 5
```

```
      □←D←A[,3]
¯4

      ρD
1

      A[⍳0]
BL

      A[1 1 2 2 4 4]
1.6 1.6 5 5 2 2
```

```
      □←F←A[2 3ρ5 3 2 4 1 7]

  7 ¯4 5
  2 1.6 ¯0.5
```

R←*A*[*B*] (L)

V H

⊢ (ρ*R*) = ρ*B*

⊢ (ρρ*R*) = ρρ*B*

DEFINITION AND EXAMPLES

DIMENSIONS AND RANKS

If B is elided, $\iota\rho A$ is assumed.

```
      A[⍳⍴A]
1.6  5  ¯4  2  7  3  ¯0.5  0  1

      A[]
1.6  5  ¯4  2  7  3  ¯0.5  0  1
```

```
      L←'TRAIL'

      L[3]              L[1 2 4 3 5]
A                 TRIAL
```

Matrix Left Argument

$R\leftarrow A[B;\underline{B}]$ (L)

M H H

R is the result of selecting from matrix A those components whose row position is B and whose column position is $\underline{B}$.

The semicolon separates the integers that make up an index of an array. B and $\underline{B}$ make up the two necessary parts of the right argument.

$\vdash (\rho R) = (\rho B),\rho\underline{B}$

$\vdash (\rho\rho R) = (\rho\rho B)+\rho\rho\underline{B}$

$\vdash A[B;\underline{B}] = ((\rho B),\rho\underline{B})\rho A[,B;,\underline{B}]$

```
      □←M←3 4⍴6 9 3 3 4 7 2 7 8 3 7 5
  6  9  3  3
  4  7  2  7
  8  3  7  5
```

```
      □←C←M[2;3]
2

      ⍴C
BL

      M[1 2 3;3]
3  2  7

      M[1 2;2 4]
  9  3
  7  7
```

```
      □←D←M[2;1 2 3 4]
4  7  2  7

      ⍴D
4

      M[3;2 4]
3  5

      M[1 1;4 2]
  3  9
  3  9
```

If the right argument is omitted, the entire matrix is assumed.

If ⊢ $\underline{B}$ = ⍳(⍴A)[2], it may be omitted. All columns are assumed.

```
      M[1 3;]          |      M[1 3; 1 2 3 4]
                       |
  6  9  3  3           |  6  9  3  3
  8  3  7  5           |  8  3  7  5
```

If ⊢ B = ⍳(⍴A)[1], it may be omitted. All rows are assumed.

```
      M[1 2 3;2 3]     |      M[;2 3]
                       |
  9  3                 |  9  3
  7  2                 |  7  2
  3  7                 |  3  7
```

```
      □←Q←3 3⍴1 2 3 2 3 1 3 1 2

  1  2  3
  2  3  1
  3  1  2
```

```
      □←T←Q[;Q]          'EAT'[T]

  1  2  3          |  EAT
  2  3  1          |  ATE
  3  1  2          |  TEA
                   |
  2  3  1          |  ATE
  3  1  2          |  TEA
  1  2  3          |  EAT
                   |
  3  1  2          |  TEA
  1  2  3          |  EAT
  2  3  1          |  ATE
```

Rank-3 Array Left Argument

R is the result of selecting from A those components whose positions in each coordinate of A are $\bar{B}$;B;$\underline{B}$.

B and $\underline{B}$ are the rows and columns,

respectively, of the array.

The right argument must have $^{-}1+\rho\rho A$ semicolons.

```
      □←H←2 3 4⍴⍳24
 1  2  3  4
 5  6  7  8
 9 10 11 12

13 14 15 16
17 18 19 20
21 22 23 24
```

```
      H[2;3;2]
22
```

```
      H[1;2;1 3]
5  7
```

```
      H[2;2 3;1 4]
17 20
21 24
```

```
      H[1 2;1 3;2 3]
 2  3
10 11

14 15
22 23
```

If $\vdash B = \iota(\rho A)[N]$, where B is the N th member of the index list, it may be omitted.

```
      H[;2;4]
8  20
```

```
      H[;1 3;]
 1  2  3  4
 9 10 11 12

13 14 15 16
21 22 23 24
```

```
      G←H[;1 3;2 3⍴ 4 3 1]
      ⍴G
2 2 2 3
```

Note: The result depends on the index origin (See Origin Command, page 141).

SPECIFICATION

For $A \leftarrow B$, the value of B, together with its rank and dimension(s), is assigned to A, which must be either a variable or a quad. It is the action of specification that causes a variable to be a scalar, vector, matrix, or rank-N array.

```
      N←4+9?.5    Assign the value of the expression
                  4+9?.5 to the variable named N.

      ⍴(⍳4)∘.=⍳4
4 4

      M←(⍳4)∘.=⍳4 The variable M, by specification,
                  is a matrix whose dimensions are
                  4 4.

      ⍴M
4 4

      ⍴⍴M
2
```

One or more elements of an established array may be assigned values by specification. In this case an index value within brackets following the variable name appears as the left argument of specification:

```
      ⎕←N←¯2+⍳7
¯1 0 1 2 3 4 5

      ⍴N[2 3 5]
3

      ⎕←N[2 3 5]←9 10 11
9 10 11
      N
¯1 9 10 2 11 4 5
```

```
      M←2 3⍴⍳6
      ⍴M[2;1]
BL

      M[2;1]←7
      M

  1  2  3
  7  5  6

      M[1 2;1 2]←2 2⍴9 10 11 12
      M

  9  10  3
 11  12  6
```

If the index expression is omitted from within the brackets, every component of the array is assumed:

```
      N←⍳8
      N[]←5
      N
5  5  5  5  5  5  5  5
```

If an indexed variable appears to the left of a specification arrow, the rank and dimension of the variable are unchanged:

```
      V←2 9 3.2 5
      ⍴V
4

      V[1 4]←5.1 3

      ⍴V
4

      V
5.1  9  3.2  3
```

A←*B* is an expression and has as its value, dimension(s), and rank the value, dimension(s), and rank of *B*.

Multiple Specification. The command $M\leftarrow 3\times N\leftarrow 2$ assigns the value 2 to N and the value $3\times N$ or 6 to M. Repeated use of specification in a single command is known as multiple specification. Its uses are illustrated in the examples below.

```
      E←1+D←1+C←1+B←1+A←0
      A,B,C,D,E
0  1  2  3  4

      ⎕←A←35×4
140
      A
140

      (R←8),S←9
8  9
      R
8
      S
9

      ⎕←V←⍳4
1  2  3  4
      D←V[⍳3]←0
      V
0  0  0  4
      D
0

      'THE VALUE IS ';T←6×325
THE VALUE IS 1950

      T
1950

      ,F←2 3⍴'AFGTSF'

AFGTSF

      F

AFG
TSF
```

Often a specification is used within a branch command:

```
→0, ρ⎕←'GIVE MESSAGE AND QUIT'

→6×10≥I←I+1
```

Note: In the following program

```
      S←5.3
      Z←(S←8)×S
      Z
64
```

the value of Z indicates that the expression in parentheses was evaluated first. (Otherwise Z would have had the value 42.4.) In the program,

```
      T←8.8
      Q←(T←5)×⌈T
      Q
45
```

the value of Q indicates that the expression in parentheses was not evaluated first. (Otherwise Q would have the value 25.) Since, as these examples suggest, no order is specified for determining when an expression in parentheses is evaluated, it may not be possible to predict what the value of a command will be if that value depends on when an expression in parentheses is evaluated. This problem of order arises only in commands of multiple specification in which a variable is specified in an expression to the left of where it is first used, such as $Z←(S←8)×S$. It is best to avoid writing such commands.

In general, this rule of no specified order does not affect the value of commands. For example, in the command $M←(3×⌊6.321)|37$, M has the value 1, regardless of when the expression $(3×⌊6.321)$ is evaluated. Similarly, in the command $Q←(T←3×8)÷9×2$, T has the value 24, and Q has the value 1.333333333, regardless of when the expression $(T←3×8)$ is evaluated.

EVALUATION OF EXPRESSIONS

The value of a compound expression is the result of executing all functions in the expression. Expressions in APL are always evaluated in the same way. Reading from left to right, each function or specification symbol operates on the entire expression to the right of it, up to the right parenthesis of the pair that enclose it:

`63÷3+7`	The quotient of 63 divided by the value of the expression 3 plus 7
`31⌊A×⍴B`	The minimum of 31 and the value of the expression *A* times the value of the expression `⍴B`

As a result of the rule that each function operates on the expression to the right of it, the rightmost expression is evaluated first, then the next rightmost, and so forth, until the entire expression has been evaluated. In other words, the order of execution is from right to left.# For example, `63÷3+7` has the value 6.3, not 28. Below are some more expressions and their values.

Expression	Value
`3+4×5+6`	47
`2×⍳4`	2 4 6 8
`⍳4×2`	1 2 3 4 5 6 7 8

For:
U: `¯2 4 6 1 3 9`
V: `1 5 3 2 6 3`
W: `2.4 6.7 5.3 ¯2.7 ¯8.1`

Expression	Value
`V[6 1 4]×⌈/V`	18 6 12
`⍴U,⍴V`	7
`⌈2.7÷⌊/U`	`¯1`

#A discussion of the choice of the right-to-left execution will be found in K. E. Iverson, Elementary Functions (Science Research Associates, 1966), Appendix A.

Parentheses. Parentheses in an expression indicate departures from the order of execution otherwise determined by the structure of an expression. The sequence indicated by parentheses is followed. Left and right parentheses must be properly paired.

Expression	Value
`(3+4)×5+6`	77
`(⍳4)×2`	2 4 6 8
`(⍴U),⍴V`	6 6
`(V-U)[(V>U)/⍳⍴V]`	3 1 1 3
`(+/(⍳5)*2)*.5`	7.416198487

Structure of an Expression. Between any two constants, variables, or quads (ignoring parentheses) there must be one, and only one, dyadic function:

Valid	Invalid
`4×B`	`4B`
`4+⍳B`	`A∈∨B`
`(5+A)÷9`	`(7*A)B`

If a function symbol represents either a monadic or a dyadic function, the monadic is assumed if the symbol to its immediate left represents a function; in `9|*⌊A`, the functions `*` and `⌊` are monadic; `|` is dyadic.

Quad and Quote-Quad in Expressions

For Output. If a quad ⎕ or quote-quad ⍞ appears to the immediate left of a specification symbol, output is specified; that is, the value of the expression will be displayed. The quad, unlike a variable, retains no value:

```
      ⎕←4×6÷7×3
1.142857143
      ⎕
⎕: 
```

(See Input, page 71)

Note: The left-hand quad and specification symbol may be elided (see also Multiple Specification, page 66):

```
      4×6÷7×3
1.142857143
```

Judicious use of the quad is helpful for seeing intermediate results in a compound expression--for example, the following sort of vector X:

```
      X←9 12 ¯8 51 4 ¯2 33
      X[(+/X∘.≥X)⍳⍳⍴X]
¯8 ¯2 4 9 12 33 51
```

How this expression builds up the sort can be seen by inserting ⎕←'s at key points in the expression:

```
      X[⎕←(⎕←+/⎕←X∘.≥X)⍳⍳⍴X]
1 0 1 0 1 1 0
1 1 1 0 1 1 0
0 0 1 0 0 0 0
1 1 1 1 1 1 1      X∘.≥X
0 0 1 0 1 1 0
0 0 1 0 0 1 0
1 1 1 0 1 1 1
4 5 1 7 3 2 6      +/X∘.≥X
3 6 5 1 2 7 4      (X∘.≥X)⍳⍳⍴X
¯8 ¯2 4 9 12 33 51 X[(+/X∘.≥X)⍳⍳⍴X]
```

For Input. If a quad or quote-quad appears anywhere except to the immediate left of a specification symbol, execution of the expression is interrupted until an expression to replace ⍞ or a character string to replace ⍵ is entered.

A ⍞ to the right of a specification arrow can be replaced by any expression, numbers or characters, or a system command. When an expression containing a ⍞ is executed, the symbol ⍞: is printed to indicate a request for input:

```
      A←365×⍞
⍞:
      25
      A
9125

      →⍞
⍞:
      5    Command 5 is the next
           command executed.

      B←⍞
⍞:
      'HELLO'

      B
HELLO

      →((A←⍞)∈1 2 3)/5
⍞:
      5
      A
5
```

Note: No branch occurs since the value of (5∈1 2 3)/5 is the empty vector.

Simply entering a carrier return or spaces and a carrier return will cause ⍞: to reappear. Entering)*CLEAR*,)*LOAD*,)*OFF* or)*OFF HOLD*, or)*CONTINUE* or)*CONTINUE HOLD* to replace ⍞ changes the contents of the active workspace, and so terminates the request for input.

If other system commands are entered to replace □, the expression containing the input quad is not affected and □: will reappear after the command is executed.)*SAVE* or)*CONTINUE* or)*CONTINUE HOLD* entered to replace □ will cause the workspace to be saved in the □ state. When that workspace is loaded, □: will appear after the save report. This can be used to put a message in a workspace as follows:

```
      'BEWARE!  THIS WS HAS ORIGIN 0',0⍴□
□:
      )SAVE TRYOUT
14.11.12  12/12/99
□:
      1    (Any number or variable can
            be entered.)
BEWARE!  THIS WS HAS ORIGIN 0

      )LOAD TRYOUT
SAVED 14.11.12 12/12/99
□:
      2
BEWARE!  THIS WS HAS ORIGIN 0
```

The state indicator will contain a □ if)*SI* or)*SIV* is entered to replace □:

```
            3×□
□:
            )SI
□
□:
            5
15
```

It is not possible to edit a function in the □ state. Entering → will terminate request for input.

A ⍞ to the right of a specification arrow takes everything keyed in as character data. When an expression containing ⍞ is executed, the keyboard unlocks with the carrier at the left margin and no symbol is printed. Use of ⍞ eliminates the need to

enclose character strings in quotes or to double quotes within strings. Replacing ⍞ with a string of characters yields a vector; a single character yields a scalar; and no character yields the empty vector.

```
      A←⍞
No symbol appears; the input starts
at the left margin.


TABLE
      A
TABLE

      A←⍞
'TABLE'

      A
'TABLE'

      Q←⍞
DON'T
      Q
DON'T
      ⍴Q
5

      T←⍞
M
      ⍴T
BL

      X←⍞
a carrier return
      ⍴X
0

      B←⍞
6+7×9÷4
      B
6×7×9÷4

      A←'MISSISSIPPI'∈⍞
MISSOURI
      A
1  1  1  1  1  1  1  1  0  0  1
```

Entering *O* backspace *U* backspace *T* (⍬) will terminate a request for character input, and in a function has the same effect as →.

Comments

The symbol ⍝ (○ overstruck with ∘) at the beginning of a line of type signals that the line is a comment line and is not to be executed. Any valid characters may be used in the comment text:

```
      ⍝ P. BROWN MATH101  4/12/99
```

Each line of comment must begin with ⍝:

```
      ⍝ A DEMONSTRATION OF THE
      ⍝ USE OF MULTIPLE LINE
      ⍝ COMMENTS
```

A comment used in a function definition has a line number:

```
     ∇ Z←CLOSE S;R
[1]   ⍝ DELETES MULTIPLE SPACES
[2]   R←S≠' '
[3]   Z←(R∨1⌽R)/S
     ∇
```

Function definition cannot be closed on a comment line, since the ∇ is regarded simply as a character in the comment.

DEFINED FUNCTIONS

APL provides a rich set of primitive functions (see Tables I through V); nevertheless there is always a need to supplement the set of primitive functions with specially designed functions. Defined functions meet this need. A sequence of commands that a user stores for use at his convenience is called a defined function.

The symbol for a defined function is an identifier and is called the function name. The syntactic form and the name of the function are established in the function header. The remaining part of a defined function is the function body, the program or sequence of commands constituting the function rule.

Defining a Function

A ∇ (called "del") preceding a function name declares a change from execution mode to function definition mode. A second ∇ terminates function definition.

In definition mode, no execution of APL commands occurs, and no errors other than character errors, definition errors, and label errors are reported. Instead each command is stored as part of the definition. The system commands *)SAVE*, *)COPY*, *)PCOPY*, and *)CONTINUE* cannot be executed during function definition, and *)ERASE* of the function being defined or edited will not be executed. All other system commands may be executed during definition. This does not affect the numbering of commands.

After the syntax (of the function) has been established in the header, the body of the function definition begins. The line number 1 is printed by the computer within brackets [1], signifying that the first line of the function program may be entered. Each line thereafter, subject to editing, is numbered consecutively.

Examples of the six types of defined functions appear in Table VI (page 81).

Header Types

Defined functions may have zero, one, or two arguments and zero or one result. Thus there are six types of syntax for defined functions. The three types that have an explicit result as indicated in the header correspond to monadic (one argument) primitive functions, dyadic (two argument) primitive functions, or constants (no argument). Defined functions with explicit results may appear in compound expressions. The remaining three types of defined functions are defined without explicit results. These must necessarily appear alone in a command; they cannot appear in a compound expression, except as the last function to be executed.

The header types are illustrated in Fig. 2. Examples of each type of function will be found in Table VI on page 81.

	EXPLICIT RESULT	NO EXPLICIT RESULT
MONADIC	∇Z←*NAME Y*	∇*NAME Y*
DYADIC	∇Z←*X NAME Y*	∇*X NAME Y*
NILADIC	∇Z←*NAME*	∇*NAME*

Fig. 2 Header Types

Dummy Variables. Recall that the syntax of the primitive functions in the section on definitions (pages 20 to 64) was symbolized as *R*←m *B* for monadic functions and as *R*←*A* d *B* for dyadic functions. The letters *A*, *B*, and *R* were used simply to show the position of the arguments and the existence of a single result. Before a function could be used in an expression, however, the letters *A* and *B* would have to be replaced by expressions which have value. The letters *A*, *B*, and *R* are dummy variables; they serve as placeholders.

The dummy variables in the header of a defined function indicate the syntax of the function. For example, ∇*T*←*Q HYP P* sets up the function *HYP* to be a dyadic function with an explicit result. Once *HYP* is defined, it can appear in a command such as *H*←3 *HYP* 4. The syntax of the command follows the syntax established in the function header. The dummy variables appearing in any one header must all be distinct, for example, the header ∇*H*←*F H* is invalid.

Just as the primitive functions were defined in Tables I through V in terms of dummy variables *A*, *B*, and *R*, so a defined function's rule is written in terms of its dummy variables. If, for instance, the function *HYP* calculates the hypotenuse of a right triangle, the function rule might be *T*←((*Q**2)+*P**2)*.5. As you can see, it is written in terms of arguments *Q* and *P* and result *T*. In the evaluation of the expression 3 *HYP* 4, *Q* in the function rule is replaced by 3, and *P* in the function rule is replaced by 4. *T* is displayed as the result of executing *HYP*.

If *T*, *P*, and *Q* had no values assigned to them prior to the execution of *HYP*, they will have no values after the function has been executed. If *T*, *P*, and *Q* had values assigned to them prior to the execution of the function, *T*, *P*, and *Q* will have those same values after the function has been executed. In other words, the use of *T*, *P*, and *Q* as dummy variables does not affect their use as variables outside the function.

Local Variables. It is convenient to make the values of variables that have no relevance or use outside the defining function--such as variables used for counters--available only within the defining function. Such variables--called local variables--have value only in a particular function, in contrast to global variables, whose values are always accessible except in certain instances of dynamic localization (see page 79) and in certain cases of suspended execution (see Suspension of Execution, page 98).

Variables that are to be local to a function are declared so in the function header at the time the function name and syntax are established. The header type is followed by the list of local variables, each preceded by a semicolon. For example, the header ∇*Z*←*P Q*;*I*;*J*;*K* establishes variables *I*, *J*, and *K* as local variables.

If a function *F* contains a local variable *A*, the global variable *A* or the function *A* cannot be used within the function *F*. Except for the above restriction, a local variable can have the same name as a global variable, a function, or a variable local to some other function. During function execution the local variable is always dominant. After a

function is executed, any value assigned to a local variable is lost and the variable resumes the definition it had prior to the execution of the function. That is, if the variable had no value prior to execution of the function, it has no value after execution. If the variable had value prior to execution, it has the same value after execution.

Function execution is entered with values assigned to dummy arguments, but not to local variables. Values for local variables are assigned within the function; otherwise there is no difference in the behavior of dummy variables and local variables. Generally, when a variable is called local, it is assumed that it may be either a dummy variable or a local variable.

```
      A←235                          A, a global variable
      N←'TEST'                       has value 235. N, also
                                     a global variable, has
                                     value 'TEST'. No
                                     values are assigned to
                                     Z and S.

    ∇ Z←N CANTOR S;A                 A dyadic function
[1]   A←'0123456789'                 with local variables
[2]   Z←(N,N+1)⍴S                    A, Z, N, and S.
[3]   Z←1 1↓Z[;1+⍳N]                 Note specification
[4]   Z←'.',A[10|1+A⍳Z]              of A within function
    ∇                                body.

      4 CANTOR '.1435.9278.0836.6104'
.2345

      A                              A, N, S, and Z
235                                  resume condition
                                     prior to execution
      N                              of the function.
TEST
                                     Recall that no value
      S                              has been assigned to
VALUE ERROR                          S or Z, local
      S                              variables.
      ^

      Z
VALUE ERROR
      Z
      ^
```

Dynamic Localization (Block Structure). The local variables of one function are accessible in other functions invoked by that function. Suppose that a monadic function *F* with argument *X* has local variables *I* and *J*. If function *F* invokes another function *G*, the local variables *I* and *J* of *F* and the argument *X* are accessible in *G* (or in a function *H* invoked by *G*) as long as *G* (or *H*) does not have its own local variables *I*, *J*, or *X*. Furthermore, any specifications of *I*, *J*, or *X* in *G* (or *H*) will be specifications to *I*, *J*, and *X* as local variables.

In the example below, the function *PRIME* stores *X* prime numbers in the global variable *P*. Function *LIMTEST* tests for a limit as *N* approaches infinity of +/(⍳*N*)÷*PRIME N*. *LIMTEST* which invokes *PRIME* has a local variable *P* that shadows the global *P* of *PRIME*. The reason that *P* was made local to *LIMTEST* was to avoid filling up the workspace with an unneeded vector. *P* was not made local to *PRIME* to preserve previously computed prime numbers if *PRIME* is not invoked by a function that has a local variable *P*.

```
     ∇ R←PRIME X
[1]    →(X≤⍴P)/6
[2]    R←¯1↑P
[3]    →(0=P|R←R+2)/3
[4]    P←P,R
[5]    →1
[6]    R←X⍴P
     ∇

       P←2 3
       PRIME 5
2 3 5 7 11

       P
2 3 5 7 11

     ∇ LIMTEST R;N;P
[1]    N←⍴P←2 3
[2]    N;' ';+/(⍳N)÷PRIME N
[3]    →(R≥N←N+1)/2
     ∇

       P←'TEST'
       LIMTEST 5
2 1.166666667
3 1.766666667
4 2.338095238
5 2.792640693

       P
TEST
```

Another example is shown below. The function *SIMULATION* finds the velocities of a mass at certain intervals as it is brought to a stop by one of two types of spring. Argument *P* is a four-component vector made up of type of spring, initial velocity, mass, and delta *X*. Because of dynamic localization, *P* is local to both *COMPRESSION* and *FINT*, which are invoked by *SIMULATION*, and variable *X* of *SIMULATION* is also local to *FINT*:

```
    ∇ V←SIMULATION P;X
[1]   X←0,P[4]×⍳⌊COMPRESSION÷P[4]
[2]   V←((P[2]*2)-2×FINT÷P[3])*0.5
[3]   V←(2,⍴V)⍴X,V
      ∇

    ∇ Z←COMPRESSION
[1]   ⍝HOW FAR MASS COMPRESSES SPRING
[2]   →2+P[1]
[3]   →0,Z←P[3]÷18÷P[2]*2
[4]   →0,Z←¯3+(9+P[3]×P[2]*2)*0.5
      ∇

    ∇ Z←FINT
[1]   ⍝INTEGRAL OF FORCE FUNCTION OVER DISTANCE
[2]   →2+P[1]
[3]   →0,Z←9×X
[4]   Z←X×3+X×0.5
      ∇
```

WITH EXPLICIT RESULTS

MONADIC ∇Z←*NAME Y*

```
     ∇ R←SORT V
[1]    R←V[⍋V]
     ∇

       SORT 4 ¯2 8 12 ¯9 6
¯9 ¯2  4  6  8  12

     ∇ Z←MAGICSQ X;M
[1]    →(0≠2|X)/3
[2]    →0,ρ⎕←'NOT ODD ORDER'
[3]    M←(X,X)ρ⍳X*2
[4]    Z←(⌈X÷2)⌽(¯1+⍳X)⌽[1](¯1+⍳X)⌽M
     ∇

       MAGICSQ 3

  8  1  6
  3  5  7
  4  9  2
```

DYADIC ∇Z←*X NAME Y*

```
     ∇ L←KEY CODE MSG
[1]    L←(PUN,KEY⌽ALPHA)[(PUN,ALPHA)⍳MSG]
     ∇

       PUN←'.,! '
       ALPHA←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

       ¯12 CODE 'YQQF YQ MF YUPZUSTF'
MEET ME AT MIDNIGHT

     ∇ I←A WHEREIN B
[1]    I←(∧/[1](¯1+⍳⍴A)⌽A∘.=B)⍳1
     ∇

       'ANU' WHEREIN 'JANUARY'
2
```

NILADIC ∇Z←*NAME*

```
      ∇ Z←PI
[1]   Z←○1
      ∇

      PI
3.141592654


      ∇ M←DATE
[1]   M←'/0123456789'[1+1 1 0 1 1 0 1 1\1+(6⍴10)⊤25]
      ∇

      DATE
07/18/68
```

WITH NO EXPLICIT RESULT

MONADIC ∇*NAME Y*

```
      ∇ EXPAND TEXT
[1]   T←((2×⍴TEXT)⍴1 0)\TEXT
[2]   T[2×⍳¯1+⍴TEXT]←'*'
[3]   T
      ∇

      EXPAND 'HYMAN KAPLAN'
H*Y*M*A*N* *K*A*P*L*A*N


      ∇ ENDZERO N;A
[1]   A←+/⌊N∘.÷5*⍳⌊5⊛N
[2]   A;' CONSECUTIVE TERMINAL ZEROS IN !';N
      ∇

      ENDZERO 10
2 CONSECUTIVE TERMINAL ZEROS IN !10
      !10
3628800
```

DYADIC *X NAME Y*

```
      ∇ WEIGHT PER ATOMS;P
[1]   C←100×P÷+/P←WEIGHT×ATOMS
      ∇

      1.008 15.994 PER 2 1
      C
11.19378123  88.80621877
```

```
     ∇ BASE CONVERT NUMBER
[1]    ALPHA[(((⌊1+BASE⍟NUMBER)⍴BASE)⊤NUMBER]
     ∇

       ALPHA←'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
       )ORIGIN 0
WAS 1
       16 CONVERT 255
FF
```

NILADIC ∇*NAME*

```
     ∇ ADD
[1]    C←A∧B
[2]    →(C[1]=1)/6
[3]    A←A≠B
[4]    B←1⌽C
[5]    →1×0=∧/~C
[6]    'ERROR'
     ∇

       A←1 1 0 0 1
       B←0 0 1 0 1
       ADD
       A
1 1 1 1 0

     ∇ PROBLEMS;T
[1]    T←3 2 2⍴?12⍴10
[2]    T[;;0]←1⌈T[;;0]
[3]    ANSW←+/T+.×10 1
[4]    Q←1 1 0\[1]'0123456789'[T]
[5]    Q[;2;]←'¯'
[6]    Q←0 0 1 1\Q
[7]    Q[;1;0]←'+'
     ∇

       )ORIGIN 0
WAS 1
       PROBLEMS
       Q

  55
+ 39


  42
+ 19


  15
+ 32


       ANSW
94 61 47
```

Function Editing

Replacements, deletions, and insertions of lines in a defined function can be made as shown in Table VII. The function header may also be edited. If it is, a syntax check will be made. An example of function editing follows Table VII.

Table VII--Function Editing

Assume *F* is a defined function containing 10 commands.
The character C stands for any command.
A blank line indicates indentation for input.

IF YOU WANT TO--	AND LAST OUTPUT WAS--	YOU ENTER--	COMPUTER RESPONDS--
Enter definition mode to add more commands		∇*F*	[11]
Override a line number	[4]	[7] C	[8]
Insert a command	[4] [6.2]	[7.1] C [3.11] C ∇*F*[.9] C	[7.2] [3.12] [1]
Enter definition mode to change a command and remain in definition mode		∇*F*[6] C	[7]
Enter definition mode to change a command and leave definition mode		∇*F*[6] C∇	
Delete a command	[7]	1050: linefeed 2741: attn (Any other sequence will not delete it.)	[8]

IF YOU WANT TO--	AND LAST OUTPUT WAS--	YOU ENTER--	COMPUTER RESPONDS--
Display a function definition and stay in definition mode	[8]	∇*F*[□] [□]	∇ *F* (The entire function) ∇ [11]
Display a function definition and leave definition mode	[5]	∇*F*[□]∇ [□]∇	∇ *F* (The entire function) ∇
Display a command and change it	[5]	∇*F*[6□] [6□]	[6] C [6]
Display a command and leave definition mode.	[7]	∇*F*[6□]∇ [6□]∇	[6] C
Display the function beginning with command *N*	[8]	∇*F*[□*N*] [□*N*]	[*N*] C (Commands from *N* on) ∇ [11]
Display function header and change it	[10]	∇*F*[0□] [0□]	[0] Header [0]
Leave definition mode	[6]	∇	
Erase a function (or group or variable)		)*ERASE F*	

Note: Keying in a ∇ after a command causes an exit from definition mode. All lines are renumbered as consecutive integers after exit from definition mode. To stop a function display, signal attention (see page 123).

An example of function definition is shown below.

```
      ∇Z←F
[1]   Z←(+/N)÷     ⍴N
[2]   [.5] 'THE AVERAGE IS'      Inserting a command.
[0.6] [⎕]                        Request for display.

    ∇  Z←F                       The display:
[0.5] 'THE AVERAGE IS'           Lines are arranged
[1]   Z←(+/N)÷⍴N                 in numerical order.
    ∇
[2]   [0]Z←F N∇                  Change header and
                                 close of definition.

      ∇F[⎕]∇                     Request for display.

    ∇ Z←F N                      The display:
[1]   'THE AVERAGE IS'           Lines were renumbered
[2]   Z←(+/N)÷⍴N                 to consecutive
    ∇                            integers when the
                                 definition was closed.
```

Line Editing. Replacements, deletions, and insertions of characters in a one-line command or in the function header can also be made during function editing by overriding the present line number with [*N*⎕*K*], where *N* is the number of the line to be edited or 0 for the function header and *K*, the approximate place (by number of spaces from the left margin) for editing to begin. Line *N* or the header is displayed, and the carrier returns to the next line and spaces to the *K*th column from the left.

1. To delete any number of characters, key in a slash beneath each character to be deleted:

```
      ∇F
[6]   [3⎕10]
[3]   -C×R×T÷5+4*6
            /
```

The line is displayed with each character underscored by a / deleted and closed -*C*×*R*×÷5+4*6. The carrier waits at the end of the command; an addition to the command may be made before entering:

```
[3]   -C×R×÷5+4*6÷9×⍳3
                   ‾‾‾
                   an addition
```

2. To insert a character or characters between two adjacent characters *X* and *Y*, type a single digit below character *Y* indicating the number (from 1 to 9) of blanks to be inserted to the left of *Y*:

```
[6]   [3⎕10]
[3]   -C×R×T÷5+4*6
              2
```

The line is displayed with two blanks between *T*

and ÷. The carrier waits at the leftmost blank for an insertion:

```
[3]   -C×R×T   ÷5+4*6
          waits here
        ←7   insertion
```

Line 3 looks like this:

```
[3]   -C×R×T←7÷5+4*6
```

Typing an *A* below a character will insert 5 spaces between that character and the one to its left. A *B* will insert 10 spaces, a *C*, 15 spaces, and so forth, in multiples of 5.

If *ALPHA* is the alphabet and *L* the letter, the number of spaces inserted is $5\times ALPHA\iota L$. What letters of the alphabet are possible depends on how much blank space remains on the line (width minus columns used). If *SR* is the space remaining and is greater than or equal to 5, the possible letters are *A* through *ALPHA*[⌊*SR*÷5].

If you make a mistake in typing the insertion, the backspace-attention (linefeed) procedure ordinarily used for correcting errors will erase the entire line above and to the right of the correction point:

```
[6]  [3□10]
[3]  -C×R×T÷5
           2
[3]  -C×R×  T÷5
         7+    insertion
          ^      erasure
          ×     correction
[4]  [3□]
[3]  -C×R×7×
```

If you have not allowed enough room for the insertion and you overstrike, you will get a character error or an overstruck character.

3. To <u>replace</u> a character with another, put a slash below the character and a digit or a letter to

the right of the slash:

```
      ∇F
[6]   [3⎕10]
[3]   -C×R×T÷5+4*6
            /A

[3]   -C×R×T↓_____5+4*6
          waits here
          PS÷1.    insertion
```

Line 3 looks like this:

```
[3]   -C×R×TPS÷1.5+4*6
```

Line numbers may also be edited in the ways described above. So, for example, if you want to duplicate the command currently at line 3 to a position following line 5, you would do the following:

```
      ∇F[3⎕2]
[3]   A+B×C÷2.3
 /3
[↓___]   A+B+C÷2.3
 waits here
 5.1   insertion
```

Line 5.1 looks like this:

```
[5.1]  A+B×C÷2.3
```

Line 3 is still part of the function definition; it is unchanged and should be deleted if it is no longer required.

4. To add to the end of a line, enter anything but a slash, a number, or a letter. The line is

displayed unaltered, and the carrier waits at the end of the line:

```
[6]   [0□10]
[0]   F←A B;I;J;K
                          a carrier return
[0]   F←A B;I;J;K ↵
                      waits here
                  ;L;M  addition
```

The header looks like this:

```
∇F←A B;I;J;K;L;M
```

5. To quit line editing, either signal attention while the line is printing out or, if the line has already been printed, cause a character error:

```
[6]   [3□10]
[3]   -C×R×T
                  ⍷
CHARACTER ERROR
                  ^
[3]   [3□]
[3]   -C×R×T
```

A *DEFN ERROR* report will be given if editing the header makes it syntactically incorrect or if altering the function name makes it the same as that of another object in the workspace.

Branching

Branching is used generally within a defined function to direct the execution of commands. It is denoted by the symbol → followed by an expression E, →E. Line 4 of the function *SQROOT* shown below is a branch on condition--if ∧/,*R*=*S*, the function terminates; otherwise line 5 is executed; line 6 is an unconditional branch--to line 3:

```
     ∇ R←SQROOT N;S
[1]    N←|N
[2]    R←1
[3]    S←.5×R+N÷R
[4]    →(∧/,R=S)/0
[5]    R←S
[6]    →3
     ∇
```

The value of E, the expression to the right of the branch arrow, determines the number of the line, if any, that is to be executed next.

1. If the value of 1↑E is within the range of line numbers of the function being executed, the next line executed is line 1↑E:

```
[7]   →5
```
Line 5 is the next executed.

```
[7]   →START+1
```
The line whose number is the value of *START*+1 is the next executed.

2. If the value of 1↑E is outside the range of line numbers, the execution of the function terminates:

```
[7]   →0
```
There is no line 0, so execution of the function terminates.

3. If the value of E is the empty vector, no branch occurs and the next line executed is the line that immediately follows. If there is no line, this is the end of the execution:

```
[7]   →(T=5)/3
```
If the value of *T* is 5, line 3 is the next

executed. If it is not, line 8 is the next executed, if it exists; otherwise execution of the function terminates.

E may be scalar or vector. The scalar or the first component of a vector must be a positive integer or zero. A label (see Labels, page 94) is the only symbol that may appear to the left of a branch arrow. Fig. 3 below shows some examples of branch commands.

Branch to *A* or execute next line:

```
→(X r Y)/A
→(X r Y)ρA
 →A×⍳X r Y
```

Branch to line *A*1 or line *A*2:

```
→(A1,A2)[1+X r Y]
→((X r Y),~X r Y)/A1,A2
```

Branch to one of several lines:

```
→((X r Y),(X r Y),X r Y)/A1,A2,A3
```

→*I*⌽*V*, where *V* is a vector of line numbers and *I* is a counter.

r: < ≤ = ≥ > ≠ ∨ ∧ ⍱ ⍲ ∈

Fig. 3 Examples of Branch Commands

If the branch arrow is used with no argument → , execution of the current function *F* terminates along with the entire sequence invoking *F*. For example, if *G* invokes *F* , execution of both *F* and *G* will terminate if → is a command in *F*. If *H* invokes *G*, which invokes *F*, execution of *F*, *G*, and *H* will terminate if → is a command in *F*. (See also State Indicator, page 99.)

Branching, →E or →, is also used to direct the action to be taken if the execution of a function has been suspended (see Suspension of Function Execution, page 98).

Branching--Affected by Function Editing. All lines are renumbered as consecutive integers after exit from definition mode, and so branching to a line number is often affected by the insertion or deletion of commands. For example, if a function containing the command →3 has a command inserted between lines 2 and 3, what had been line 3 becomes line 4 at the close of function definition.

Labels. One remedy is to use labels with commands which are 'branch to' points. A command label is established by preceding the command with a variable name. A colon separates the label from the command:

```
[2]   START:N←N+1
```

The value of the label is the number of the line with which it is associated at the close of function definition. Labels are respecified each time function definition is closed:

```
      START
2

      ∇F[1.5]a command∇

      ∇F[□]∇
    ∇  F
[1]   a command
[2]   a command
[3]   START:N←N+1
[4]   a command
[5]   →(N<10)/START
[6]   ...
    ∇

      START
3
```

Labels are local constants which are defined when the function is executed. The value of the label cannot be respecified in a command. An attempt to do so will result in a *SYNTAX ERROR*. Like local

variables labels are assessible in functions invoking a function containing labels (see Block Structure, page 79). If a function containing labels is suspended and if editing affects the value of the label, an *SI DAMAGE* report will be given and it will be necessary to edit the function in a nonsuspended state.

Recursive Function. A function that invokes itself in the body of its definition is recursive. For example, the function *FAC* below produces the factorial of its argument. Observe line 2:

```
     ∇ Z←FAC N
[1]    →4×⍳N=0
[2]    Z←N×FAC N-1
[3]    →0
[4]    Z←1
     ∇
```

Function *FIB* produces the first *N* terms of a Fibonacci series, whose first two terms are *A*. Observe line 3:

```
     ∇ R←N FIB A
[1]    R←A
[2]    →(N=2)/0
[3]    R←(N-1) FIB A
[4]    R←R, (R+¯1⌽R)[⍴R]
     ∇
```

Function *PR* produces a matrix of all the permutations of order *M*. Observe line 2:

```
     ∇ P←PR M;Z
[1]    →2×M>,P←1 1⍴1
[2]    P←PR M-1
[3]    P←⍉(1 0 +.⌽⍴P)⍴(,⍉P),(1↑⍴P)⍴M
[4]    Z←,⍉((1↑⍴P),M)⍴¯1-⍳M
[5]    P←Z⌽((M,1)×⍴P)⍴P
     ∇
```

Tracing the Execution of a Function

It is often useful to have the values of some (or all) commands of a function *F* typed out as execution of the function progresses. To accomplish this, a trace on the function may be set by specifying a value *V* for the trace control *T*Δ*F*. *V* may be an integer or a vector of integers, *T*Δ*F*←*V*. Only the components in *V* whose values correspond to line numbers of *F* are significant. For each such component *F*[*C*] and the value of the command *C* is printed. If *F*[*C*] is a branch command, the value of the expression to the right of the branch arrow is printed. *T*Δ*F*←0 or *T*Δ*F*←⍳0 discontinues the trace.

```
     ∇ B←PASCAL N
[1]    B←1
[2]    B←(B,0)+0,B
[3]    →2×N>B[2]
     ∇

       TΔPASCAL←2 3

       M←PASCAL 3
PASCAL[2] 1  1
PASCAL[3] 2
PASCAL[2] 1  2  1
PASCAL[3] 2
PASCAL[2] 1  3  3  1
PASCAL[3] 0

       TΔPASCAL←0
       M←PASCAL 3
       M
1  3  3  1
```

The trace vector may be specified on a line of a function: if, for example, *T*Δ*F*←9×*I*<*J* is a command in *F*, command 9 will be traced as long as the value of *I* is less than the value of *J*.

The trace vector is not a variable: it does not appear in the variable list; it cannot be examined; and it cannot be copied. A trace cannot be set for a nonexistent function. Deleting a function that has a trace control set for it also deletes the trace control vector. Editing a line removes the trace control for that line.

Stop Control

A function stop is a planned suspension of the execution of a function. It is established by setting a stop control vector in the same way that a function trace is established by setting the trace control vector (see Tracing, page 96).

*S*Δ*F* is the stop control for function *F*; *S*Δ*F*←*V* specifies the stop control vector where *V* is an integer or vector of integers. Only the components in *V* whose values correspond to command numbers of *F* are significant. For each such component *C*, execution of the function is stopped just before command *C*, there is a linefeed, *F*[*C*] is printed, there is a linefeed, and the keyboard unlocks. The function is now in normal suspended execution (see Suspension, page 98). Execution of the function can be terminated or resumed by appropriate branching. *S*Δ*F*←0 or *S*Δ*F*←⍳0 discontinues the stop control.

An example is shown below for a function named *TRIP*.

```
      S∆TRIP←7  15
      TRIP

TRIP[7]
       suspension activities
      →7

TRIP[15]
```

Like the trace control vector, the stop control vector can be used within a defined function--to suspend execution after a certain number of iterations in a loop, for example. The stop control is not a variable and cannot be examined or copied. A stop control cannot be set for a nonexistent function. Deleting a function that has a stop control set for it also deletes the stop control vector. Editing a line removes the stop control for that line.

Suspension of Function Execution

The execution of a function may be stopped before completion because a stop control for the function had been set (see Stop Control, page 97), because an attention had been signaled (see Attention, page 123), because an error had been detected (see Errors in a Defined Function, page 106), or because the APL operator has sent a *PA* message (see page 143). In any case, when a suspension occurs, the name of the suspended function and the line number of the next command to have been executed is typed out.

When the keyboard unlocks after a function suspension, the computer is in execution mode. Anything that can normally be done in execution mode can be done during function suspension (except for the restrictions on function editing discussed on page 99). As long as a function is suspended, the local variables of that function are active and can be examined.

A branch to the line number listed in the suspension report will resume execution of the function at that line. A branch to any other number will resume execution of the function at that command. As usual, branching to a number outside the range of line numbers of a function terminates execution of the function.

It is usually not good practice to execute a function that is already in a suspended state. If a suspended function is executed again, the execution of the function begins at another level within the level on which the function was suspended (see State Indicator, page 99).

State Indicator. The system command)*SI*, called the state indicator, causes a type-out of all functions that are currently active, with the most recent first:

```
      )SI
R[3]  *
T[7]
F[1]  *
```

The symbol * after the function name indicates that the function is in suspended execution. No letter after the function name indicates an active, but not suspended, function. Such a function is called pendant. A function may be pendant because it called another function. The number in brackets following the function name is the number of the next line to be executed. A function that is pendant cannot be erased.

An attempt to edit a pendant function will produce the report *DEFN ERROR*. Suppose the state indicator lists the following active functions:

```
      )SI
T[5]  *
G[5]  *
F[3]  *
P[7]
T[5]  *
G[3]
M[2]
```

Functions *T* and *F* could be edited, but not *M*, *G*, and *P*. Note that function *G* is both pendant and suspended. Since in one of its states *G* is not suspended, it cannot be edited. Notice also that function *T* has been suspended twice.

Clearing the state indicator is done by terminating execution of each suspended function on the active list. This can be done by entering one branch arrow → for each * in the list. The expression ⍴⍳27 tells how many items there are in the

state indicator. (See Copy Command, page 138, for another procedure for clearing the state indicator.)

If no functions are active,)*SI* will produce no report.

The system command)*SIV* prints for each active function, the line number of the next command to be executed, an *, if required, and a list of the local variables and labels declared in the function header:

```
        )SIV
R[3]    *     I    J
T[7]    *     M
F[1]    *     J    S
G[3]          A    TR    PS    N
J[4]
```

In this example the variable *J* local to function *R* is dominant, and the variable *J* local to function *F* and the function *J* are inaccessible (shadowed).

Locking Functions

A function can be locked, or protected, by opening or closing the function definition with a ⍢ (∇ overstruck with ~) instead of ∇:

```
      ∇F          |       ⍢F          |       ⍢F
[6]   ⍢     [6]   ∇     [6]   ⍢
```

A locked function can only be executed, copied, or erased; it cannot be revised or displayed in any way. The faulty command is not displayed if an error occurs in the execution of a locked function. The trace control and stop control for a function cannot be changed after the function is locked.

ERRORS

Execution of a command is terminated as soon as an error is detected, and any partial result is lost. The error is classified (*SYNTAX*, *VALUE*, *RANK*, and so forth), the command in which the error appears is displayed, and a caret, ^, marks the approximate place the error was detected:

```
      (⍳5)+⍳4
LENGTH ERROR
      (⍳5)+⍳4
      ^
```

If there is more than one error in a command, only the first one found by the computer will be signaled at the first error report. If, after correcting the error indicated, the command is reentered, the next error will be signaled:

```
      4B+6÷0
DOMAIN ERROR
      4 B+6÷0
        ^

      4B+6÷1
SYNTAX ERROR
      4 B+6÷1
        ^
```

If an error occurs during the execution of a command that has multiple specification, specifications to the right of the caret will have taken place:

```
      R←45
      T←¯8×R←.5
DOMAIN ERROR
      T←¯8×R←0.5
          ^
      R
0.5
```

Table VIII, which follows, is a table of error reports.

Table VIII--Errors

ERROR	CAUSE	CORRECTIVE ACTION	SEE PAGES
CHARACTER	Improper overstrike	Reenter command.	
DEPTH	Excessive use of defined functions within defined functions	Attempt repeated execution of →. If this fails, save, clear, copy.	99, 107 136, 148
DEFN	Improper attempt at function definition or syntax of function header improper as a result of header editing or function is pendant.	A function or variable already has the name. Delete it or rename the function you are trying to define. If function name is an indexed identifier, rename the function. During interrupted function execution; clear the state indicator by repeated branches →. It is not possible to edit locked functions.	75,91 99 101
DOMAIN	Function not defined for given values of the arguments	Reformulate command.	20-63
INDEX	Request for nonexistent element of an array	Reformulate command.	13

ERROR	CAUSE	CORRECTIVE ACTION	SEE PAGES
LENGTH	Arrays not conformable because combining coordinates have unequal lengths.	Reformulate command.	20-63
RANK	Function not defined for array(s) of this structure	Reformulate command.	20-63
SI DAMAGE	Attempt to edit label lines of a suspended function	Clear state indicator	99
SYMBOL TABLE FULL	Too many names used.	Save, clear, copy. Try again. If same report, erase some functions, groups, and variables; then save, clear, and copy.	
SYNTAX	Ill-formed command	Reformulate command.	69
SYSTEM	Indeterminate problem internal to the machine	No action necessary. A clear workspace is loaded automatically. All prior work in workspace is wiped out, as if *)CLEAR* had been executed.	

ERROR	CAUSE	CORRECTIVE ACTION	SEE PAGES
VALUE	Value for this variable not previously specified or the dummy result of a defined function that has an explicit result not specified during execution of the function	Specify a value for the indicated variable.	64,75
WS FULL	Workspace overloaded	Delete objects no longer required, clear the state indicator, or use equivalent formulation that uses less space.	148

Errors in a Defined Function

During function definition, character errors, labeling errors, and definition errors are detected. No other errors are detected until the commands containing them are executed. When an error is detected during function execution, execution is suspended at the offending command (see Suspension, page 98), the type of error is indicated (see Table VIII, page 103), and the function name, line number of the offending command, and the command are displayed:

```
     ∇ A F B          The function as
[1]    N←1            defined.  Note error
[2]    N←N+1          in line 3.
[3]    C←A⁻B
[4]    etc.
     ∇

       3 F 4          Execution of F
SYNTAX ERROR
F[3]  C←A⁻B           Error detection
          ^
```

An error can be corrected either while in suspended execution or after terminating function execution.

1. Correcting Errors After Terminating Function Execution. A branch with no argument will cause an exit from the last suspended function on the active list. A similar branch for each suspended function on the active list is necessary to terminate that function's execution:

```
SYNTAX ERROR
F[3]  C←A⁻B
          ^

       )SI
F[3] *

       →
       )SI
```

Once execution of a suspended function has been terminated, the function can be edited as necessary and reexecuted:

```
      ∇F[3] C←A-B∇
      3 F 4
```

2. Correcting Errors in Suspended Execution. The function can be edited following the usual editing procedures as long as the function is not pendant, that is, it does not appear in the state indicator without an * (see page 99).

```
SYNTAX ERROR
F[3]  C←A¯B
          ^

      )SI                    state indicator
F[3] *

      ∇F[3]C←A-B∇            correction
```

After editing, to resume execution of the function at the point at which it was suspended, execute a branch to the line number (in the above example, →3). It is not good practice to invoke the function again while it is suspended.

If the lines containing labels are edited in suspended execution, the report *SI DAMAGE* might result. The *SI DAMAGE* report may also result if the local variables in the function header are changed. In such cases exit from function execution is necessary before editing the function.

Some unanticipated things that may happen for which no error report is forthcoming.

IF--
you get no apparent answer when you enter an expression and the carrier is at the left margin,

TRY--
entering a quote.

REASON:
Character strings are marked with enclosing quotes. You may have an open quote.

IF--
a defined function takes an inordinate time to execute,

TRY--
signaling attention and checking the branching in the function.

REASON:
Execution of an endless loop.

IF--
nothing happens for a long time,

TRY--
entering.

REASON:
Perhaps you forgot to.

IF--
a defined function or an expression does not return the anticipated answer, and you've checked it thoroughly,

TRY--
)SIV to see whether some functions are active, and what their local variables are.

REASON:
Some identifiers, used in one function on the active list as local variables, may be shadowing other variables or functions.

IF--
something seems wrong with the branching within a function,

TRY--
opening and closing function definition.

REASON:
Labels may have been respecified. Entering and closing function definition will respecify labels to coincide with line numbers.

IF--
when a defined function that uses ? is executed, the chain of random numbers generated is always the same,

TRY--
saving the workspace containing the function after each use of the function, or changing the seed by using the function *SETLINK* in 1 *WSFNS*.

REASON:
The random seed (see page 116) has not been modified since the last time the workspace was saved.

IF--
after signaling attention the carrier does not space over for normal input,

TRY--
entering until it does.

REASON:
The terminal may be out of phase with the computer.

IF--
execution of a function is suspended with no error, stop control, or attention,

TRY--
)SI and normal procedures for resuming execution.

REASON:
A signal was transmitted (possibly because of interference on the telephone lines) that was interpreted by the computer as an attention signal.

USING *APL\360*

This section discusses procedures for using the IBM System/360 version of APL (usually called *APL\360*).

The terminal#--an IBM 1050 Tele-Processing System, an IBM 2740 Communications Terminal, or an IBM 2741 Communications Terminal--is the typewriterlike device used for communicating with the computer. See Fig. 4 below for switch settings and Table X, page 112 for some terminal procedures.

	Switch	Position
2741 and 2740	LCL/COM	COM
	ON/OFF	ON
1050	Power	On
	System	Attend
	Printer 1	Send/Rec
	Keyboard	On
	EOB	Manual
	System Disc	Up
	Test	Off

The positions of any other switches on the terminal face, if provided, are irrelevant.

The 1050 terminals have a 'line control' switch, usually in the rear section of the 1051 control unit. This switch must be on. Additional control keys should never be used unless you understand their functions.

After turning the power on, depress any key whose adjacent light is on. If any light is on, depress RESET key also.

Fig. 4 Turning on the Switches

#See K. E. Iverson and A. D. Falkoff, APL\360 User's Manual (International Business Machines Corporation, 1968), Part 1, Gaining Access for required and optional features.

Terminal

Keyboard.

APL\360 KEYBOARD

The APL keyboard is similar to a standard keyboard. Some significant features of the keyboard are indicated below.

1. There is one set of alphabetic characters (italic capitals). These characters are in the unshifted position (the shift key is not depressed) of the keyboard and occupy the same keys that they do on a standard typewriter.

2. The period, comma, and numerals are also in the unshifted position of the keyboard and occupy the same keys that they do on a standard typewriter.

3. The numeral 1 is the first character of the top row.

4. The negative sign, ¯, is in the shifted position (the shift key is depressed) of the 2 key.

5. The arithmetic function symbols + - × ÷ are on the two keys in the upper right-hand corner.

6. Most of the other symbols are in the shifted position on the keyboard.

In many instances there is a mnemonic connection between the function symbol and the letter whose key it shares. For example, members of the pairs ∈ and *E*, ⍳ and *I*, and ○ and *O* look somewhat alike. Other mnemonic aids are ? and *Q* for query, ⍴ and *R* for rho, ~ and *T* for tilde, * and *P* for power, ' and *K* for quote, ⊥ and *B* for base, and | and *M* for magnitude.

Table X--Terminal Procedures

ENTER	To indicate that you have completed entering a line of data. 2741: RETURN key. 2740: RETURN key and EOT key. 1050: RETURN key and then depress the ALT CODING and 5 keys simultaneously.
ATTENTION	To interrupt whatever the computer might be doing (see page 123). 2741 with interrupt feature: Depress ATTN key. If this does not work, use procedure described below. 2741 without interrupt feature, 2740, and 1050: Remove DATA-phone receiver from cradle. Depress talk button for 3 to 10 seconds, and then depress data button. Modem (hard-wired): Turn power off and on quickly.
CORRECTIONS TO A LINE BEFORE ENTERING	To correct errors detected before the line has been entered. Backspace to error and strike ATTN key (2741), INDEX key (2740), or LINEFEED key (1050); all symbols at and to the right of the carrier position are deleted. Corrections are entered on new line. 1050 only: A line can be canceled before entering by depressing ALT CODING key and 0 key simultaneously, then entering. The DATA CHECK and RESEND lights will come on; depress DATA CHECK and RESEND keys. Retype command.
CREATING OVER-STRUCK CHARACTERS	Hit one key, backspace, and hit the other key; for example, o backspace | creates ϕ. The order of striking is immaterial (see also page 154).
1050 ONLY: DATA CHECK AND RESEND LIGHTS GO ON	Depress DATA CHECk and RESEND keys. Command will have to be retyped.
2740 AND 2741: *RESEND* REPORT	When an error occurs in transmission from a terminal to the computer, the word *RESEND* is printed. The carrier returns, and the keyboard unlocks so that the line can be retyped.

Communicating with the Computer

Establishing a Connection with a DATA-phone on a Dial-up Terminal.

1. Remove receiver from cradle.

2. Depress talk button.

3. Dial the appropriate number.

   a) If you get a busy signal, hang up and try another number.

   b) If you get no answer after a few rings, hang up and try another number.

   c) If you hear a high-pitched steady tone, press data button. The data button should light and will remain lit as long as the connection lasts.

4. After the keyboard is unlocked, enter a right parenthesis followed by your account number and associated key (see Locks and Keys, page 115). The keyboard of a 1050 is unlocked when the proceed light goes on; the keyboard of a 2740 or 2741 is unlocked if depressing the shift key causes the typing element to rotate. A full sign-on dialog appears below.

```
      )999999:KEY
OPR:  SYSTEM COMING DOWN AT 12 CST.
021)  16.33.05  08/15/99 SPAKIN

      A P L \ 3 6 0     S.R.A.  -- CRIS CENTER

SAVED  12.12.01  08/14/99
```

999999 is the sign-on number.

KEY is the key (password).

OPR: MESSAGE is an optional message from the APL operator. The message generally contains schedules, information about the system, and so forth.

021) is the port number. (The port number is the number of the access port on the computer that connects your terminal to the computer.)

16.33.05 is the sign-on time (local time on the computer) in hours, minutes, and seconds.

08/15/99 is the current date.

SPAKIN is the user's identification.

APL\360 is the name of the program and system. *S.R.A. -- CRIS CENTER* is the location of the computer.

SAVED 12.12.01 08/14/99 If the last time you terminated a work session on the computer an unlocked active workspace was saved in *CONTINUE*--by)*CONTINUE*,)*CONTINUE HOLD*, or a disconnect--*CONTINUE* will automatically be loaded and a *SAVED* report given. (See also Continue, page 132). If there is no *SAVED* report, the active workspace is *CLEAR WS*.

See Table XI, page 145, for corrective actions to take in case you get a trouble report when you attempt to sign on.

Directions for establishing a connection with a modem line (hard-wired directly to the computer) or a leased line should be obtained from the installation. Directions for establishing a connection with an acoustic coupler can be obtained from the manufacturer's instructions. The sign-on procedure (step 4, page 113) is the same.

Locks and Keys. An account number can be protected from unauthorized use by requiring a user-specified password as well as the account number for sign-on. A password is established at sign-off by following the ending command-- *)OFF*, *)OFF HOLD*, *)CONTINUE*, or *)CONTINUE HOLD*-- with a colon and a code up to eight letters in length. This locks the account number. Thereafter at sign-on, the account number followed by a colon and the password serve as a key to gain access to the system.

```
      )OFF:SECRET
off report

      )999999:SECRET
on report
```

A password can be changed by following the sign-off command with a colon and the new password. The password can be discontinued by following the ending command with a colon.

A workspace can similarly be protected from unauthorized use with a password. This is discussed on page 129.

Active Workspace. After you have signed on, you are in execution mode and are able to communicate with the computer. That is, you can give it commands (input) and get responses from it (output). All such communication takes place in a block of space in the computer's storage area known as the active workspace. This is the environment that a user works in. A clear active workspace has the following features:

1. Storage space. The number of bytes (see page 148) of storage in each workspace is preset at a fixed value for a given system. (To find out the size of a clear workspace,)*CLEAR* and ⎕22. See also, System Information, page 148.)

2. Line width of 120 spaces. (This can be changed. See Width Command, page 142.)

3. Origin 1. (This can be changed. See Origin Command, page 141.)

4. Display of 10 significant digits, retention of 17. (The display can be changed. See Digits Command, page 142.)

5. A fuzz (see page 120) of approximately `1.0E¯13`.

6. Random Seed. This is a number (`16807`, i.e., `7*5`) used as the start point for generating the result of the random function. Each use of *?* modifies the seed. Saving a workspace causes the current seed to be saved.

7. The name *CLEAR WS*.

Input Mechanics. You can enter input whenever the keyboard is unlocked. (The first five sections of the manual describe types of APL input. Other types of input are discussed in System Commands, page 125.)

1. Generally the carrier spaces six places before the keyboard is unlocked to permit input. The exceptions are (1) after a quote-quad (see page 72) where input begins at the left margin; (2) the editing of a command in a defined function, where you specify the number of places that the carrier is to space (see page 88); and (3) after a *RESEND*, where the carrier remains at the left margin.

2. Backspacing to insert additional symbols is permitted. The expression evaluated is the expression that appears on the paper just prior to entering regardless of how the expression was constructed. A good rule of thumb is "what you see goes in." So, for example, if you key in `6+7` backspace backspace backspace backspace backspace `2*`, you see on the paper `2*6+7` and the expression that the computer evaluates is `2*6+7`.

3. If a 1050 terminal is provided with a standard black-and-red ribbon, input is printed in red and output in black.

4. Multiple spaces in a numeric expression or in a system command can always be shrunk to one space. Inserting extra spaces in an expression generally does not change the meaning (value) of the expression. Inserting or deleting a space will change the meaning of the expression in the following cases:

   a) The meaning changes if a space is inserted within a constant or a variable, with the exception of inserting extra spaces between successive components of numeric vector constants:

      MEAN is not the same as *ME AN*.

      'ABC', *'AB C'*, and *'AB C'* are all

different.

465 is not the same as 4 65.

b) The meaning changes if deleting the space between a defined function name and its arguments makes the result look like a valid identifier (<u>see</u> Identifiers, page 16):

F 2 and *F*2 are not the same.

F .2+4 and *F*.2+4 are the same.

c) The meaning changes if the space between adjacent components of a vector constant is deleted:

453 and 45 3 and 4 5 3 and 4 53 are all different.

Output. The display of all data, with the exception of numeric arrays of rank two or greater, begins at the left margin.

1. A fractional number is displayed with one leading zero, whether or not it was entered that way:

```
      .5          |       000.3       |       ÷3
0.5               | 0.3               | 0.3333333333
```

A fractional number is not displayed with trailing zeros:

```
          .5000   |         3.000
    0.5           | 3
```

2. If a scalar or a component of a vector is less than $1E^{-}5$, greater than $1EN$ (where N is the number of digits displayed), or an integer greater than $^{-}1+2*31$, display of that number will be in exponential notation--regardless of which form the number was entered in. Exponential notation is always displayed as a number whose magnitude is less than 10 but not less than 1, immediately followed by *E*, immediately followed by an integer:

```
      3E4           |       3E¯2
30000               | 0.03

      10000000000   |       .00000002634
1E10                | 2.634E¯8

      5E10          |       ¯12E11
5E10                | ¯1.2E12
```

3. The number of significant digits retained by the system may be less than the number of significant digits keyed in:

```
      A←1234567891234567899
      B←1234567891234567811
      A-B
0     Manually executed, A-B
      would have value 88.
```

4. The number of significant digits displayed is less than the number retained by the system:

```
      □←A←1111111111111111
1.111111111E15
      B←1.111111111E15
      A-B
111111
```

5. Fuzz. Comparisons between numbers are relative. If two numbers are equal within a certain tolerance, the relationship will be considered true. This tolerance is called fuzz and is approximately 1.0E⁻13. The results of floor and ceiling, the relational functions, and functions that use the relational functions as a basis for determining the result may be affected by fuzz.

```
      2.111111111111111=2.1111111111111119
1

      2.111111111111111≤2.1111111111111119
1
      2.111111111111111-2.1111111111111119
¯4.440892099E¯16

      ⌈6+10*-⍳16
7  7  7  7  7  7  7  7  7  7  7  7  6  6  6  6

      X←⍳8
      X[6.00000000000004]
6
```

6. Numeric vectors are displayed with spaces between components. Character vectors are displayed with no spaces between components:

```
      ¯3+⍳5
¯2 ¯1  0  1  2

      'ABCDEF'
ABCDEF

      'ABC  EFG'
ABC  EFG
```

7. A matrix is displayed as a rectangular arrangement of its components. A rank-N array is displayed as a set of matrices. (In array *H* for $2\geq\rho\rho H$, there are $\times/\bar{}2\downarrow\rho H$ matrices, whose dimensions are $\bar{}2\uparrow\rho H$.)

   Numeric matrices and rank-N arrays are indented two spaces. Character matrices and rank-N arrays are displayed with no spaces between columns:

```
      3 4⍴⍳12

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

      2 2 3⍴⍳12

 1  2  3
 4  5  6

 7  8  9
10 11 12

      3 4⍴'DOWNRIPETORN'

DOWN
RIPE
TORN

      2 2 3⍴'TEAEATATEYUM'

TEA
EAT

ATE
YUM
```

8. The empty vector--a vector of no components--can be entered in several ways--⍴0 or 0⍴2 or ⍳0 or ''. (The empty vector is also called the null vector.) Besides the empty vector, there are empty arrays of rank 2 or more. One or more of the components of the dimension vector of an empty array are zero. The expressions 0 3⍴5 and 0 0⍴0 and 0 1 3⍴2 and 3 0⍴5 are examples of distinct empty arrays.

   The following example shows the display of the empty vector and an empty array.

Empty Vector	Empty Array
⍳0	0 3⍴10

Paper advances leaving a blank line (denoted by BL in this manual).

Note: Although there is no visible distinction between the empty vector and an empty array, the distinction does exist.

The expression '' represents an empty character vector, and the expression ⍳0 represents an empty numeric vector. In general, these two expressions can be used interchangeably. The only time a distinction can be seen is when either is the right argument of expansion:

0\''	0\⍳0
BL	0

If an empty array is an argument of a scalar function, the result will be an empty array.

3+⍳0	0≠0 3⍴4
BL	BL

The empty vector is used in branching (see page 92), to initialize a vector, or to print a blank line during the execution of a function. These three uses are shown in the function *STAT* below (lines 1, 4, and 6).

```
     ∇ STAT X;R
[1]    R←⍳0
[2]    R←R,(X=⌊/X)/X
[3]    X←(X≠⌊/X)/X
[4]    →2×⍳0≠⍴X
[5]    'MEDIAN: ';.5×+/R[⌈.5×0 1+⍴R]
[6]    ''
[7]    'AVERAGE: ';(+/R)÷⍴R
[8]    ⍝LINES 1 THROUGH 4 SORT X
     ∇
```

9. If the display of a vector or of a row of a matrix or rank-N array exceeds one line (see Width Command, page 142), the excess is printed on the next line with an indention of six spaces.

10. A display can be stopped at any time by signaling attention (see page 112). If attention is used during execution, output stops as soon as the attention is received. The paper linefeeds, and the carrier returns to the left-hand margin and indents six spaces for new input. An attention will not interrupt the execution of most system commands, but it will stop the display of a report.

    If attention is used to halt the display of a function definition, the computer will leave definition mode if a del closed the display command. If no del closed the display command, the computer will remain in definition mode and the command number *N*+1 will appear; *N* is the last command of the function.

    The attention can be used to suspend the execution of a function. But suspending a function by attention, unlike suspending it by the stop control vector (see page 97), cannot be regulated. With the stop control vector, you specify the exact place you want a function suspension to occur. With attention, it is not possible to know which command is being executed at the time you signal attention. Output is not a reliable guide, since the execution of the function may be well in advance of the output. In fact, the results of more than one command may be ready and waiting to be displayed, and so an attention will not only halt the current output but will also wipe out any data waiting to be displayed. Furthermore, since the time it takes to display a line is considerably longer than the compute time, the computer might be finished executing a function long before the output is completed. Because of this, attention used during the execution of a function may either suspend execution of the function or return the function to another level of execution. If execution of a function is suspended by an attention, the function name and line number are typed out (see Suspension, page 98).

If the function definition contains a right-hand quad or quote-quad, using attention may cause another request for input. When the request for numeric input is given, entering → will terminate the function execution. A request for character input can be terminated by typing *O* overstrike *U* overstrike *T*, in that order.

An attention signal will not interrupt the execution of a command. If it becomes desirable to interrupt execution within a command, signal attention a second time. The report *INTERRUPT* will be given and the interrupted command will be displayed with a caret marking the approximate place that the line was interrupted.

System Commands

System commands# are used for workspace control--affecting the state of the active workspace; communications--transmitting messages around terminals; inquiry--providing information about the active workspace; library--affecting the state of the library; and signing on and off. All system commands, and only system commands, have as their first character a right parenthesis. System commands cannot be used in APL expressions and cannot be part of a function definition, and conversely APL expressions cannot be used in system commands. The following paragraphs describe the system commands. Table XI, page 145, lists the trouble reports that may occur, the associated problem, and the corrective action to be taken. A summary of all system commands will be found in Appendix B, page 150.

Function List Command. The command *)FNS* lists alphabetically the names of all defined functions in the active workspace:

```
      )FNS
COS     STOP    TRS     L
```

If *)FNS* is followed by a letter, all function names from that letter on will be listed:

```
      )FNS T
TRS     L
```

Variable List Command. The command *)VARS* lists alphabetically the names of all global variables in the active workspace:

```
      )VARS
C       CRP     FN      PRM     Q
```

If *)VARS* is followed by a letter, all variable names

#See also APL\360 User's Manual, Part 2, System Commands

from that letter on will be listed:

```
      )VARS M
PRM     Q
```

Group List Command. The command)*GRPS* lists alphabetically the names of all groups in the active workspace:

```
      )GRPS
LESSON1 LESSON2 LESSON3 STAT
```

If)*GRPS* is followed by a letter, all group names from that letter on will be listed:

```
      )GRPS R
STAT
```

Group Membership Command. The command)*GRP* NAME lists the names in the group NAME:

```
      )GRP LESSON1
GAS    P      PLOT     VS
```

Libraries and the Library Command. Assigned to each user identification number is a private library in which a user may save workspaces. Each user has a quota of workspaces. He may have a workspace named *CONTINUE* (see page 132) in addition to this quota.

The command *)LIB* lists the names, but not the keys, of the saved workspaces in a user's own private library:

```
      )LIB
APPLECORE
MANUAL
CONTINUE
PROBSOLV
```

Library numbers 1 through 999 are reserved for public libraries. All users have access to the workspaces in public libraries. For example, anyone who wants to can use the workspaces in public library 1, which is distributed with the system. (Each of the workspaces in library 1 has a descriptive function, usually named *DESCRIBE*, that describes how the functions in that workspace are used.) Workspaces in public libraries are established and maintained by individual users (see Save, page 128).

If the command *)LIB* is followed by a public library number, the workspaces in that public library will be listed:

```
      )LIB 1
ADVANCEDEX
PLOTFORMAT
APLCOURSE
WSFNS
TYPEDRILL
NEWS
```

Saving and Loading Workspaces. The save and load procedures for storing work in library workspaces and for bringing saved work into the active workspace is illustrated in the following examples (see also diagram, page 149).

```
      )CLEAR
CLEAR WS
```

Command discussed on page 133.

```
      )LIB
MYWORK
CONTINUE
```

To check workspace names. User has one named workspace in addition to *CONTINUE*.

APL activities: function definition, execution of commands, variables specified, and so forth.

```
      )SAVE MYOTHERWORK
SAVED  10.51.01  01/23/99
```

Save command fixes *MYOTHERWORK* as a library workspace. The name may be any valid identifier, although only the first 11 characters are recognized--that is, two names that differ only at the twelfth character will not establish two workspaces. A duplicate of all functions, variables, and groups as well as the width, origin, digits, fuzz, random seed, and functions active on various levels is placed in the library workspace named *MYOTHERWORK*.

```
      )WSID
MYOTHERWORK
```

Command discussed on page 133.

When a copy of the active workspace is stored, the active workspace assumes the stored workspace name and identification number (called WSID).

```
      )LIB
MYWORK
CONTINUE
MYOTHERWORK
```

APL activities.

```
      )SAVE
11.45.23  01/23/99  MYOTHERWORK
```

Replaces work in *MYOTHERWORK* with a duplicate of the active workspace. If no WSID follows the)*SAVE*, the WSID of the active workspace is assumed.

```
      )LOAD MYWORK
SAVED 11.12.51  11/20/99
```

Replaces work in active workspace with a duplicate of everything in library workspace *MYWORK*.

APL activities.

```
      )SAVE MYOTHERWORK
NOT SAVED, THIS WS IS MYWORK
```

Save command not honored. Tried to save in an already existing workspace when the WSID was different.

```
      )SAVE MYWORK:KEY
SAVED  15.23.34  01/23/99
```

Following the workspace name with a colon and a password (up to 8 letters long) locks the workspace.)*SAVE* NAME with no lock, discontinues protection.)*SAVE* retains the lock.

APL activities.

```
      )SAVE 259 EXAMPLES
 SAVED 15.10.51  03/19/99
```

The active workspace is saved in public library 259 under the name *EXAMPLES*.

```
      )LIB
MYWORK
CONTINUE
MYOTHERWORK
```

A public library workspace is not listed in)*LIB*. Nevertheless user has three workspaces plus *CONTINUE* in which to store work.

```
      )LOAD MYWORK
IMPROPER LIBRARY REFERENCE

      ) LOAD MYWORK:KEY
SAVED  15.23.34  01/23/99

      )LOAD EXAMPLES
WS NOT FOUND
```

Although user established public workspace *EXAMPLES*, it is not stored in his private library but rather is stored in the public library.

```
      )LOAD 259 EXAMPLES
SAVED  15.10.51  03/19/99

      )LOAD 234123 HISWORK
SAVED 12.12.43  09/15/99
```

Loading a workspace from another user. Library identification number (and key) is required. Identification number is optional for workspace in user's private library.

APL activities.

```
      )SAVE 234123 HISWORK
IMPROPER LIBRARY REFERENCE
```

Save command not honored. Cannot save in another user's private library.

```
      )LOAD 1 NEWS
SAVED 14.51.09  08/14/99
```

APL activities.

```
      )SAVE 1 NEWS
IMPROPER LIBRARY REFERENCE
```

Only the user who established the public library workspace can save or drop it.

A save to library workspace X causes a duplicate of everything in the active workspace to supersede everything in the library workspace X. And a load from workspace X causes a duplicate of everything in workspace X to supersede everything in the active workspace. (See Copy command, page 136, for bringing selected objects into the active workspace.)

Workspace *CONTINUE*. The workspace named *CONTINUE* is established when)*SAVE CONTINUE*,)*CONTINUE*, or)*CONTINUE HOLD* is executed or when a disconnect occurs. *CONTINUE* receives the active workspace automatically if the connection drops or if the APL operator disconnects the user (called "bouncing"). If a disconnect occurs during function definition, the definition is closed and *CONTINUE* is saved in execution mode. If a disconnect occurs during command execution, the effect is the same as ATTN and)*CONTINUE*. If the computer itself fails, the active workspace is usually not saved in *CONTINUE*.

CONTINUE can also be used as any other named workspace. It can be saved, loaded, dropped, copied from, et cetera. However, it is advisable to use it only for temporarily holding a workspace--for instance, during an involved sequence of loading, saving, and copying--since any equipment malfunction may cause an automatic save of the current active workspace.

The command)*SAVE CONTINUE* will be honored even if *CONTINUE* had not previously been loaded into the active workspace. You will never get the report *NOT SAVED*, *THIS WS IS WSID* when you execute)*SAVE CONTINUE*.

```
      )LOAD MYWORK
SAVED  15.23.35 01/23/99
```

APL activities.

```
      )FNS
COS      PLOT    TR        UVW
```

Trouble. Line drops.

```
      )999999:KEY
on report
SAVED  01.14.15  01/24/99
```

CONTINUE is automatically loaded if terminating the last work session caused *CONTINUE* to be saved--unless the active workspace was protected. If the active workspace was protected, *CONTINUE* is saved with the same lock and is not automatically loaded.

```
      )WSID     Command discussed on page 133.
CONTINUE

      )FNS
COS     PLOT     TR       UVW
```

All work in the active workspace at time of malfunction is in *CONTINUE*.

Clear Command. The command)*CLEAR* makes the active workspace a fresh, clean workspace named *CLEAR WS*.

```
      I22
6095

      )CLEAR
CLEAR WS

      I22
31872
```

Workspace Identification. The command)*WSID* returns the name assigned to the active workspace and the identification number if it is different from the sign-on number. The name and number of the active workspace is commonly called WSID (pronounced "whiz-id").

```
      )WSID
CONTINUE

      )LOAD 1 NEWS
SAVED  15.31.45  12/31/99
      )WSID
1 NEWS
```

The command)*WSID* followed by NAME changes the name of the active workspace to NAME:

```
      )SAVE MYOTHERWORK
NOT SAVED, THIS WS IS CONTINUE

      )WSID MYOTHERWORK
WAS CONTINUE

      )SAVE
14.05.14  8/15/99  MYOTHERWORK
```

The password is neither changed nor erased by)*WSID* NAME.

Dropping a Workspace. A drop command removes a workspace and its contents from the library. Use of)*DROP* is illustrated below.

```
      )LIB
MYWORK
MYOTHERWORK
CONTINUE

      )DROP MYOTHERWORK
  12.15.42  01/26/99

      )LIB
MYWORK
CONTINUE

      )SAVE NEWNAME
SAVED 12.22.04  01/26/99
```

A new workspace can be established at any time except during function definition.

```
      )DROP 23412 HISWORK
IMPROPER LIBRARY REFERENCE

      )LOAD MYWORK
WS LOCKED

      )DROP MYWORK
  12.30.21  01/26/99
```

A protected workspace can be dropped without knowing the workspace password.

Group Command. A group is a collection of names. If a name in a group is the name of a function, variable, or group in the workspace, the name is said to have a referent: it refers to something. The referent of a function name is the function definition; the referent of a variable name is a value, the referent of a group name is the group definition. The command)*GROUP* NAME LIST defines a group: NAME is the name of the group. It cannot be the same as that of a function or variable in the workspace. LIST is a list of the names that are members of the group. The members of the group may or may not have referents in the workspace.

```
      )GROUP LESSON1 PLOT VS P GAS1 V T
      )GRPS
LESSON1
      )FNS
PLOT   VS      GAS1
      )VARS
P
```

V and *T* are names in group *LESSON1* although at this time neither has a referent.

```
      )GROUP LESSON1 LESSON1 GAS2
      )GRP LESSON1
PLOT    VS      P       GAS1     V       T        GAS2
```

Repeating the name of the group in a)*GROUP* command adds to the members of the group.

The command)*GROUP* NAME disperses the group NAME. It deletes the definition of the group, but does not delete the referents of the names in the group:

```
      )GROUP LESSON1
      )GRPS

      )FNS
PLOT    VS      GAS1    GAS2
      )VARS
P
```

Copy Command. The transfer of information takes place primarily from the user's terminal to his active workspace; for example, a function that a user defines at his terminal goes to his active workspace. The transfer of information can also take place from a library workspace to the active workspace. This is done by means of a copy command. There are two ways to use the copy command:

1. The copy command can be used to copy a single object--one function, one variable, one group--from a library workspace. This is done by entering the command)*COPY* WSID key NAME.

```
      )LOAD MYWORK
SAVED 15.23.34  01/23/99

      )FNS
COS     PLOT    TR      UVW

      )VARS
AB      CD      TIP     R

      )GRPS
GP1

      )COPY WORK:KEY SIN
SAVED 12.23.32  03/19/99

      )FNS
COS     PLOT    SIN     TR      UVW

      )VARS
AB      CD      TIP     R

      )GRPS
GP1
```

When a group is copied, the group name and the referents of the group, if any, are copied:

```
      )GROUP GP1 F T GP2

      )GROUP GP2 S R

      F←T←S←T←'VARIABLE'

      )SAVE CONTINUE
SAVED 12.34.54  12/12/99
```

```
      )CLEAR
CLEAR WS

      )COPY CONTINUE GP1
SAVED  12.34.54  12/12/99

      )VARS
F       T

      )GRPS
GP1     GP2

      )GRP GP2
S       R

      S
VALUE ERROR
      S
      ^

      R
VALUE ERROR
      R
      ^

      F
VARIABLE

      T
VARIABLE
```

2. The copy command can be used to copy all objects from a library workspace. This is done by entering)*COPY* WSID key. Only the functions, variables, and groups in the workspace are copied. The digits, width, origin, random seed, suspension list, trace control, and stop control are not copied.

```
      )COPY 259 CHEM
SAVED 12.13.34  07/12/99
```

If you copy from a public library workspace or from another user's workspace, the identification number (and key) must be included in the copy command.

```
      )FNS
COS     GAS1     PLOT     SIN      TR       UVW

      )VARS
AB       C        CD       TIP      R        M

      )GRPS
GP1

      )COPY MYOTHERWORK Q
OBJECT NOT FOUND
```

Q does not exist in workspace.

The definition of an object copied into an active workspace which already has an object by that name replaces the former definition (see also Protecting Copy Command, page 139).

The copy command can be used to clear the state indicator:

```
      )SI
G[3]  *
F[2]  *
G[3]  *
F[2]  *
G[3]  *
F[2]  *
G[3]  *
T[4]

      )SAVE
19.21.40  01/27/99  MYWORK

      )CLEAR
CLEAR WS

      )COPY MYWORK
SAVED  19.21.40  01/27/99

      )SI

      )WSID MYWORK
WAS CLEAR WS

      )SAVE
19.53.31  01/27/99  MYWORK
```

Protecting Copy Command. The command)*PCOPY* is like)*COPY* except that no functions, variables, or groups will be copied if the active workspace already contains objects by those names:

```
      )LOAD 21356 TABLE
SAVED  21.23.45  11/26/99

      )VARS
A       C       S       UV

      A
345

      C
7  8  9

      S
16  8  5.6

      UV
ABCD

      )CLEAR
CLEAR WS

      A←B←C←'VARIABLE'
      )COPY 21356 TABLE
SAVED  11.23.45  11/26/99

      )VARS
A       B       C       S       UV

      A
VARIABLE

      B
VARIABLE

      C
VARIABLE

      S
16  8  5.6

      UV
ABCD
```

Using)*PCOPY* to copy a group will copy only those referents which are not names of objects already existing in the active workspace. It is possible that all the referents of a group will be copied but not the group definition.

Erasing Objects. The command)*ERASE* followed by a list of names deletes the referents, if any, of those names. If a group is erased, the referents of the group will be deleted. This means that if functions and variables are referred to in the group, they will be deleted. If a group is referred to, it will be dispersed--that is, the group definition will be deleted but not the referents of the group. A pendant function cannot be erased. A function cannot be erased while it is being edited.

```
      )FNS
F1      F2      F3      F4      F5

      )VARS
V1      V2      V3      V4

      )GRPS
G1      G2      G3

      )GRP G1
F4      F1      G2      V3

      )GRP G2
F3      V1      G3

      )ERASE F2  V2  G1

      )FNS
F3      F5

      )VARS
V1      V4

      )GRPS
G3

      ∇F1
[10]   )ERASE F1
NOT ERASED:  F1

      )SI
F2[3]   *
F3[5]
      )ERASE F2 F3
NOT ERASED:  F3
      )FNS
F3      F5
```

Origin Command. In a clean workspace, indexing and functions related to indexing, such as monadic and dyadic ⍳, have origin 1. The first component of a vector, for example, has index 1, of a matrix, 1;1, and so forth. The coordinates of an array *H* are numbered 1 through ⍴⍴*H*. The system command)*ORIGIN* 0 makes the index of the first component of a vector 0, of a matrix 0;0, and so forth. The coordinates of an array *H* are numbered 0 through ¯1+⍴⍴*H*.)*ORIGIN* 1 restores the origin to 1.

The functions affected by the origin command are indexing, index generator, index of, random, grade up, and grade down. The subscript values for subscripted functions are also affected by the origin change:

```
      )ORIGIN 1
WAS 0

      ⍳5
1  2  3  4  5

      5 2 1⍳2 5
2  1
```

```
      )ORIGIN 0
WAS 1

      ⍳5
0  1  2  3  4

      5 2 1⍳2 5
1  0
```

```
      □←M←2 3⍴9 6 5 3 2 1
9  6  5
3  2  1

      +/[1]M
12  8  6

      +/[2]M
20  6
```

```
      +/[1]M
20  6

      +/[0]M
12  8  6
```

The expression ⍳1 is a quick check of origin, since it returns 0 in origin 0 and 1 in origin 1.

Note: The system-related function *ORIGIN* found in 1 *WSFNS* can also be used to change the origin.

Width Command. The width command affects output only. In a clean workspace, the width of a line of output is fixed at 120 spaces. The command)*WIDTH* N, for N between 30 and 130, will change the output width to N number of spaces:

```
      )WIDTH 31
WAS 120
      'THE QUICK BROWN FOX JUMPED OVER THE HEDGE'
THE QUICK BROWN FOX JUMPED OVER
       THE HEDGE
```

Note: The system-related function *WIDTH* found in 1 *WSFNS* can also be used to change the width.

Digits Command. The digits command affects output only. In a clean workspace, the maximum number of significant digits displayed is fixed at 10. The command)*DIGITS* N, for N between 1 and 16 will change the number of significant digits displayed to N. The digits displayed affect the display of nonintegers in exponential representation. A number equal to or greater than 1*E*N will be displayed in exponential representation.

```
      ÷3
0.3333333333

      )DIGITS 3
WAS 10
      ÷3
0.333
      1E5
1E5

      )DIGITS 16
WAS 3
      ÷3
0.3333333333333333
      100000
100000
```

Note: The system-related function *DIGITS* found in 1 *WSFNS* can also be used to change the digits.

Message Receiving and Sending. A message from the computer operator or from another terminal can be received at a terminal anytime between sign-on and sign-off whenever the keyboard is locked. If no reply is expected, a message from the operator is identified by *OPR*:. If no reply is expected, a message from another terminal is identified by PORT: (PORT is the terminal port number); if a reply is expected, an *R* immediately follows the colon.

To send a "please reply" message, use the command)*OPR MESSAGE* or)*MSG* PORT *MESSAGE*.)*OPR* or)*MSG* PORT sends your port number, a colon, and *R MESSAGE* to the operator or addressed terminal. When the message is actually transmitted, *SENT* is printed. Your keyboard remains locked either until you get a message or until you signal attention. An)*OPR* message may be sent before you have signed on. For example:

```
      )OPR IS LMB THERE? SPAKIN
SENT

OPR: NO. HE'S AT LUNCH. BACK 15 MIN.
```

The command)*OPRN MESSAGE* or)*MSGN* PORT *MESSAGE* sends your port number, a colon, and *MESSAGE* to the operator or addressed terminal. When the message is received, a *SENT* report is given and your keyboard unlocks.

If the message is directed to a port that is signed off or to a nonexistent port, your port number and message are reflected back to you:

```
      )MSG 1000 MESSAGE
021:R MESSAGE
SENT
```

A public address (*PA*) message sent by the APL operator to all signed-on users is prefixed by *PA*!:. Since a *PA* message generally contains information of immediate interest to the user, it is sent to the user's terminal as soon as possible. A *PA*, unlike an *OPR* or *MSG* message, will interrupt the execution of a function. After the message is received, execution of the function can be resumed (see Suspension of Function Execution, page 98):

```
F[14]
PA! APL GOING DOWN AT 22:30 TONIGHT.
      →14
```

Port List Command. The command)*PORTS* lists the port number and user code for each connected port. (The expression ⍴⎕TC tells how many users are connected. See also System Information, page 148.) The user code is the first three letters of the user identification name:

```
      )PORTS
21    SPA
34    KEI
45    SPA
51    ADF
```

The command)*PORT* CODE lists all port numbers associated with the given user code:

```
      )PORT SPA
21    SPA
45    SPA
```

Ending Communication. The command)*OFF* will sign you off the system and drop the DATA-phone connection. The command)*OFF HOLD* will sign you off the system and keep the DATA-phone connection for 60 seconds, permitting another user to sign on without redialing. Both)*OFF* and)*OFF HOLD* cause all the work in the active workspace to be wiped out.

After the sign-off command, the following report is given:

```
021  17.49.29  09/12/99   SPA
CONNECTED    1.13.09   TO DATE   3.14.56
CPU TIME     0.05.05   TO DATE   0.10.09
```

021 is the port number. *SPA* is the user code (not his password). *CONNECTED* stands for actual time connected. *CPU* (Central Processing Unit) *TIME* is the time it took for the computer to execute all your commands. Time is in hours, minutes, and seconds.

The commands)*CONTINUE* and)*CONTINUE HOLD* are similar to)*OFF* and)*OFF HOLD* except that the work in the active workspace is put into library workspace *CONTINUE* and if the active workspace was not locked, *CONTINUE* will be loaded automatically at the next sign-on. A save report is given after)*CONTINUE*.

Table XI--Trouble Reports

Trouble reports associated with the system commands are listed below. Most give a clear indication of the problem.

REPORT	PROBLEM	CORRECTIVE ACTION
ALREADY SIGNED ON	Work session at the terminal is in progress.	Find person using terminal or)*CONTINUE HOLD*.
IMPROPER LIBRARY REFRENCE	Attempt to)*SAVE*,)*DROP*, or)*LIB* for a private library other than that of the account number signed on with. Attempt to store workspace in a nonexistent library.	Don't.
INCORECT SIGN ON	Form of command faulty.	Enter correctly.
MESSAGE LOST	Attention was signaled before the message was sent.	Don't signal attention until after *SENT* report appears.
NOT ERASED: NAMES(S)	Function is pendant. Function is being edited.	Clear state indicator. ∇ to end definition.
NOT GROUPED, NAME IN USE	Variable or function already has that name.	Change name of group or erase conflicting object.

REPORT	PROBLEM	CORRECTIVE ACTION
NOT SAVED, THIS WS IS WSID	An established workspace cannot be saved unless the active workspace has the same name.	Change name of active workspace before saving, using)*WSID* command.
NOT SAVED, WS QUOTA USED UP	Allotted number of workspaces has already been used up.	Use workspace *CONTINUE*, drop an unused workspace, or ask to have allotment increased.
NOT WITH OPEN DEFINITION	Terminal is in definition mode.	Close the definition with ∇.
NUMBER IN USE	Someone is signed on with that number.	Find out who. Or if you turned off the power to disconnect, try again after two minutes. Otherwise notify the APL operator.
NUMBER LOCKED OUT	Authorization for use of the number has been withdrawn.	Contact person who gives authorization.
NUMBER NOT IN SYSTEM	Either the number is not in the system or the number has a lock and wrong key was used.	

REPORT	PROBLEM	CORRECTIVE ACTION
OBJECT NOT FOUND	Workspace does not contain the object, a variable, function, or group.	
SYMBOL TABLE FULL	Too many names used.	Save, clear, copy. Try again. If same report, erase some functions, groups, and variables; then save, clear, and copy.
WS FULL	Active workspace cannot contain all the material requested. A variable will be copied completely, if at all. A partially copied function will leave the active workspace in definition mode.	Save, copy, clear workspace. Or erase unneeded objects and save, clear, copy.
WS LOCKED	No key or the wrong key was used.	
WS NOT FOUND	There is no workspace with that name in this library.	

Table XII--System Information

System Information (*R*←⌶*B*)

The family of functions denoted by ⌶ (⊥ overstruck with ⍳) provides information about *APL\360*. The argument must be scalar:

⌶19 Accumulated keying time in 60ths of a second during this session. Time during which keyboard has been unlocked awaiting entries.

⌶20 Time of day in 60ths of a second.

⌶21 CPU time since sign-on in 60ths of a second.

⌶22 Remaining unused space in workspace in bytes. A byte is a unit of storage equal to 8 binary digits. A workspace with 32000 bytes has room for approximately 32000 characters or 8000 integers or 4000 mixed numbers or 256000 logical numbers or several hundred lines of function definition.

⌶23 Number of users currently signed on.

⌶24 Your sign-on time in 60ths of a second.

⌶25 Today's date. MMDDYY in base 10.

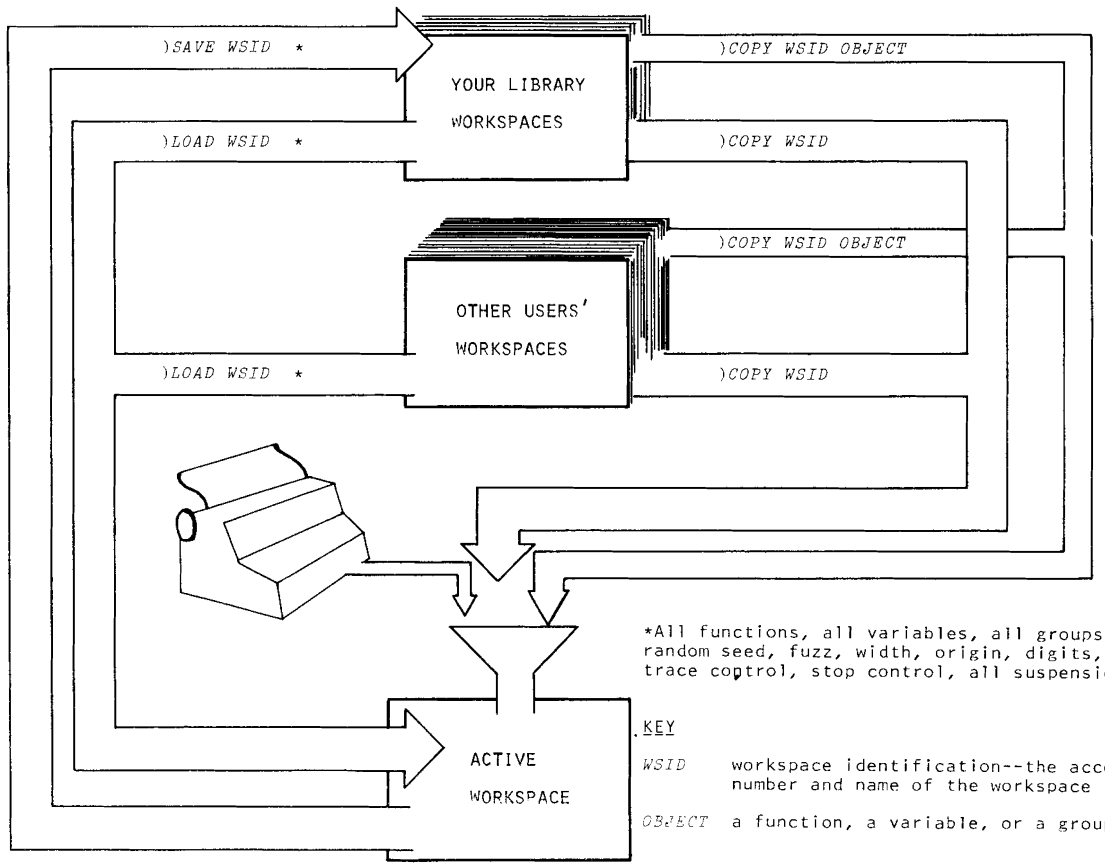
⌶26 Current value of line counter. In the execution of a defined function, this is the command number of the command being executed. ⌶26 can be used for branching. For example, →2+⌶26 is a branch to two commands beyond the present one.

⌶27 Vector of line numbers of functions in the state indicator. ⍴⌶27 tells how many items there are in the state indicator.

⌶28 The terminal device being used:
1 2741 ATS
2 2741 TSS
3 1050
4 console typewriter

⌶29 User sign-on number

Appendix A--Save-Load-Copy Diagram



Object indicated by capital roman type is a required part of the command; object indicated by lowercase roman type is an optional part of the command. Number in parentheses refers to the page on which the command is discussed.

COMMAND	PURPOSE
)CLEAR	Activate a clear workspace. (133)
)CONTINUE	Terminate a work session and store the active workspace in *CONTINUE*. (144)
)CONTINUE HOLD	Like *)CONTINUE*, but the dial-up connection is held. (144)
)COPY WSID key	Copy all functions, variables, and groups from a stored workspace. (136)
)COPY WSID key NAME	Copy a function, a variable, or a group from a stored workspace. (136)
)DIGITS 1-16	Set number of significant digits to be displayed. (142)
)DROP WSID	Delete a stored workspace. (134)
)ERASE NAME(S)	Erase objects listed. (140)
)FNS letter	List names of defined functions. (125)
)GROUP NAME LIST	Define a group NAME whose members are LIST. (135)
)GROUP NAME	Disperse group NAME. (135)
)GRP NAME	List members of group NAME. (126)
)GRPS letter	List names of groups. (126)
)LIB number	List names of workspaces in designated library. (127)
)LOAD WSID key	Activate a copy of a stored workspace. (128)
)MSG PORT text	Send text to designated port; keyboard locks. (143)
)MSGN PORT text	Like *)MSG*, but the keyboard

	unlocks. (143)
)*OFF* lock	Terminate a work session. (144)
)*OFF HOLD* lock	Like)*OFF*, but the dial-up connection is held. (144)
)*OPR* text	Send text to APL operator; keyboard locks. (143)
)*OPRN* text	Like)*OPR*, but the keyboard unlocks. (143)
)*ORIGIN* 0-1	Set index origin. (141)
)*PCOPY* WSID key	Like)*COPY*, but protecting the contents of the active workspace. (139)
)*PCOPY* WSID key NAME	Like)*COPY* NAME, but protecting the contents of the active workspace. (139)
)*PORTS*	List port number and associated user code for all terminals signed on. (143)
)*PORTS* CODE	Port number(s) associated with designated user's code. (143)
)*SAVE*	Re-store a copy of the active workspace. (128)
)*SAVE* WSID lock	Store a copy of active workspace. (128)
)*SI*	State indicator. (99)
)*SIV*	State indicator plus local variables. (100)
)*VARS* letter	List names of global variables. (125)
)NUMBER key	Sign on. (113)
)*WIDTH* 30-130	Set width of output line. (142)
)*WSID*	Identification of active workspace. (133)
)*WSID* NAME	Change identification of active workspace. (133)

Appendix C--Function Symbols

SYMBOL	NAME	PAGE	SYMBOL	NAME	PAGE
<	Less than	26	≤	Less than or equal	26
>	Greater than	26	≥	Greater than or equal	26
=	Equal	26	≠	Not equal	27
∨	Or	27	⍱	Nor	27
∧	And	27	⍲	Nand	27
-	Negation Subtraction	20 23	+	Identity Addition	20 23
÷	Reciprocal Division	20 23	×	Signum Multiplication	20 23
?	Monadic random Dyadic random	22 59	⍴	Dimension Restructuring	38 41
∈	Membership	59	~	Not	22
↑	Take	57	↓	Drop	58
⍳	Index generator Index of	37 45	○	Pi times Circular	22 24
⌽	Reversal Rotation	38 43	⍉	Monadic transposition Dyadic transposition	39 52
⍟	Natural logarithm Logarithm	21 25	*	Exponential Exponentiation	20 25
⌈	Ceiling Maximum	21 24	⌊	Floor Minimum	21 23
⍒	Grade down	40	⍋	Grade up	40
!	Factorial Combination	22 25	[]	Indexing	60
⊥	Base value	46	⊤	Representation	47
∣	Absolute value Residue	21 24	,	Ravel Catenation	37 42
/	Compression	48	\	Expansion	50
∘.d	Outer product	35	d.D	Inner product	33
d/	Reduction	30	⎕	System information	148

Appendix D--Other APL Symbols

SYMBOL	NAME	PAGE	SYMBOL	NAME	PAGE
E	Exponential notation	7	¨	Dieresis	10
¯	Negative sign	7	0...9	Digits	7
→	Branch arrow	92	←	Specification arrow	64
∇	Del	75	Δ	Delta	16
⍞	Quad	70	⍞	Quote-quad	72
'	Quote	10	∘	Small circle	35
()	Parentheses	69	[]	Brackets	60
;	Semicolon	11 60 77	.	Period Decimal point	7
:	Colon	94	_	Underscore	16
*S*Δ...	Stop control	97	*T*Δ...	Trace control	96
A...*Z* *A*...*Z*	Letters	16	∧	Caret	102
			⍝	Comment	74

Appendix E--Symbols Used in This Manual

SYMBOL	MEANING	SYMBOL	MEANING
BL	Empty array	m	Monadic function symbol
d or D	Dyadic function symbol	↔	Equivalence
⊢	Assertion	e	2.7182818284590451
π	3.141592653589793	S	Rank-0 argument
V	Rank-1 argument	M	Rank-2 argument
H	Rank-N argument	C	Any APL command
E	An expression	lim	Limit
ln	Natural logarithm	(L)	Character argument
Γ	Gamma function	β	Beta function

Appendix F--Overstruck Characters

An overstruck character is made by striking one key, backspacing, and then striking the other key.
The order in which the keys are struck is immaterial.

SYMBOL FOR	MADE WITH
Nor ⍱	∨ ~
Rotate ⌽	○ |
Reversal ⌽	○ |
Logarithm ⍟	○ *
Down Grade ⍒	∇ ⍋
Comment ⍝	∩ ○
Quote-quad ⍞	' ⎕
Nand ⍲	∧ ~
Transpose ⍉	○ \
Protected function ⍫	∇ ~
Up grade ⍏	Δ |
Factorial !	. '
Combination !	. '
I-beam ⌶	⊥ ⊤

APL\360 Reference Manual

AR HOWELL

PAKIN SRA

Reorder No. 17-1