A FORMAL DESCRIPTION OF APL

IBM PHILADELPHIA SCIENTIFIC CENTER
Data Processing Division

TECH. REPORT NO. 320-3008 NOVEMBER 1971

R. H. LATHWELL & J. E. MEZEI

PHILADELPHIA SCIENTIFIC CENTER TECHNICAL REPORTS

† 320-2900 *K. Spielberg*, An Algorithm for the Simple Plant Location Problem with Some Side Conditions, 48 p., November 1967

† 320-2901 *J. Greenstadt*, Variations on Variable-Metric Methods, 23 p., May 1967

■ 320-2902 *Y. Bard*, A Function Maximization Method with Application to Parameter Estimation, 16 p., June 1967

■ * 320-2903 *A.M. Bomberault & W.W. White*, Scheduling Empty Box Cars, 23 p., December 1966

■ 320-2904 *B.D. Gavril*, S.C.U. Special Control Unit Information Brief No. 1, 12 p., July 1967

■ 320-2905 *B.D. Gavril*, S.C.U. Special Control Unit Information Brief No. 2, 29 p., July 1967

† ■ 320-2906 *O. Gurel & L. Lapidus*, Stability Via Liapunov's Second Method, 54 p., July 1967

320-2907 *O. Gurel*, Stability Analysis of N-Dimensional Systems Through Topology of Velocity-Space, 29 p., August 1967

† ■ 320-2908 *E.S. Savas*, Computers in Urban Air Pollution Control Systems, 51 p., August 1967

† 320-2909 *J. Greenstadt*, Cell Discretization II - Discrete Equations (Elliptic Case), 27 p., August 1967

† * 320-2910 *H. Eisenpress & J. Greenstadt*, The Estimation of Non-Linear Econometric Systems, 15 p., August 1967

† ■ * 320-2911 *C.E. Lemke & K. Spielberg*, Direct Search Zero-One and Mixed Integer Programming, 62 p., September 1967

320-2912 *J. Greenstadt*, A Richocheting Gradient Method for Non-Linear Optimization, 26 p., September 1967

† ■ 320-2913 *Y. Bard*, On a Numerical Instability of Davidon-Like Methods, 4 p., September 1967

320-2914 *J. Greenstadt*, Cell Discretization III - Treatment of Discrete Equations (Hyperbolic and Parabolic Case), 13 p., September 1967

† ■ * 320-2915 *J. Rubin*, Optimal Classification into Groups - An Approach for Solving the Taxonomy Problem, 62 p., November 1967

† 320-2916 *S.G. Hahn*, On the Optimal Cutting of Defective Glass Sheets, 20 p., October 1967

† 320-2917 *R.G. Loomis, & J.J. Lorenzo*, Experiments in Mapping with a Geo Space Plotter, 18 p., September 1967

† 320-2918 *L.S. Woo*, An Algorithm for Straight Line Representation of Simple Planar Graphs, 19 p., October 1967

† 320-2919 *O. Gurel & L. Lapidus*, The Maximum Principle and Discrete Systems, 22 p., November 1967

† ■ 320-2920 *M. Guignard*, Generalized Kuhn-Tucker Conditions for Mathematical Programming in a Banach Space, 17 p., November 1967

† ■ 320-2921 *O. Gurel*, Marker Layout Problem Via Graph Theory, 42 p., January 1968

† * 320-2922 *H.P. Friedman & J. Rubin*, On Some Invariant Criteria for Grouping Data, 69 p., November 1967

† * 320-2923 *P.G. Comba*, A Language for 3-Dimensional Geometric Processing - Written Form, 40 p., December 1967

† * 320-2924 *P.G. Comba*, A Procedure for Detecting Intersections of 3-Dimensional Objects, 18 p., January 1968

■ 320-2925 *A.R. Colville*, Mathematical Programming Codes, 25 p., January 1968

320-2926 *A.R.D. Norman & M.J. Dowling*, Railroad Car Inventory: Empty Woodrack Cars on the Louisville and Nashville, 69 p., June 1967

■ 320-2927 *G. Gordon*, The Development of Simulation Languages, 34 p., January 1968

† 320-2928 *F. Freudenstein & L.S. Woo*, Kinematics of the Human Knee Joint, 30 p., January 1968

† 320-2929 *K. Spielberg*, Plant Location With Generalized Search Origin, 32 p., March 1968

† ■ 320-2930 *J.F. Raimond*, An Algorithm for the Exact Solution of the Machine Scheduling Problem, 40 p., March 1968

■ * 320-2931 *R. Shareshian & K. Spielberg*, The Mixed Integer Algorithm of N. Driebeek, 18 p., July 1966

■ * 320-2932 *R. Shareshian*, A Modification of the Mixed Integer Algorithm of N. Driebeek, 11 p., July 1966

■ * 320-2933 *Y. Bard*, Production-Transportation-Marketing Model, 22 p., October 1966

* 320-2934 *J. Greenstadt*, Further Experiments in Triangularizing Matrices, 19 p., November 1966

† 320-2935 *G. Gordon & K. Zelin*, A Simulation Study of Emergency Ambulance Service in New York City, 37 p., February 1968

320-2936 *B.D. Gavril*, S.C.U. Special Control Unit Information Brief No. 3, xx+173 p., March 1968

† ■ 320-2937 *O. Gurel & L. Lapidus*, A Guide to Methods for the Generation of Liapunov Functions, 82 p., March 1968

† ■ 320-2938 *K. Spielberg*, Enumerative Methods for Integer and Mixed Integer Programming, 115 p., March 1968

■ 320-2939 *S. Poley*, Mesh Analysis of Piping Systems, 136 p., March 1968

■ * 320-2940 *R.G. Loomis*, A Design Study on Graphics Support in a FORTRAN Environment, 33 p., May 1966

■ * 320-2941 *J. Greenstadt*, On the Problem of Fairing, 58 p., December 1966

† ■ * 320-2942 *J. Greenstadt*, On the Relative Efficiencies of Gradient Methods, 16 p., January 1967

* 320-2943 *O. Gurel & M.M. Salah*, Stability Analysis Reports, Report No. 1, 25 p., Report No. 2, 16 p., February 1967

† * 320-2944 *J. Greenstadt*, Cell Discretization I - Variational Basis, 27 p., February 1967

320-2945 *O. Gurel*, Additional Considerations on Marker Layout Problem Via Graph Theory, 34 p., April 1968

■ 320-2946 *D.W. Webber & W.W. White*, An Algorithm for Solving Large Structured Linear Programming Problems, 28 p., April 1968

† 320-2947 *L.S. Woo & F. Freudenstein*, On the Curves of Synthesis in Plane, Instantaneous Kinematics, 40 p., May 1968

† ■ 320-2948 *Y. Bard & J.L. Greenstadt*, A Modified Newton Method for Optimization with Equality Constraints, 13 p., May 1968

† ■ 320-2949 *A.R. Colville*, A Comparative Study on Nonlinear Programming Codes, 77 p., June 1968

■ 320-2950 *O. Gurel & M. Guignard*, Structure of Constrained Optimality With Respect to Higher Order Derivatives, 25 p., June 1968

■ 320-2951 *H. Salkin & K. Spielberg*, Adaptive Binary Programming, 90 p., June 1968

† ■ 320-2952 *A.M. Bomberault & W.W. White*, Networks and Transportation: The Empty Freight Car Allocation Problem, 58 p., July 1968

† * 320-2953 *L.S. Woo*, Type Synthesis in Plane Linkages, 39 p., July 1966

† ■ 320-2954 *S. Poley & C. Strauss*, A Three Dimensional Piping Design Program, 30 p., July 1968

† ■ 320-2955 *Y. Bard*, Comparison of Gradient Methods for the Solution of Nonlinear Parameter Estimation Problems, 58 p., September 1968

■ 320-2956 *H. Eisenpress & A. Bomberault*, Efficient Symbolic Differentiation using PL/I-Formac, 25 p., September 1968

† 320-2957 *O. Gurel*, Stability of the Pair (X,f), 14 p., November 1968

† 320-2958 *S. Gorenstein*, Printing Press Scheduling for Multi-Edition Periodicals, 22 p., November 1968

320-2959 *C. Hao, L. Woo, V. Vitagliano, & F. Freudenstein*, Analysis of Control-Mechanism Performance Criteria for an Above Knee-Leg Prosthesis, 51 p., December 1968

† ■ 320-2960 *E. Balas*, Project Scheduling with Resource Constraints, 23 p., January 1969

† 320-2961 *O. Gurel & L. Lapidus*, Topology of Velocity-Space for Stability Analysis, 12 p., January 1969

■ 320-2962 *J. Cord*, Generalized Upper Bounds with Upper Bounded Variables, 47 p., January 1969

IBM PHILADELPHIA SCIENTIFIC CENTER TECHNICAL REPORT NO. 320-3008 NOVEMBER 1971

A FORMAL DESCRIPTION OF APL *

R. H. LATHWELL

J. E. MEZEI

PHILADELPHIA SCIENTIFIC CENTER
IBM CORP.
3401 MARKET STREET, PHILADELPHIA, PENNSYLVANIA, 19104

This paper was prepared for COLLOQUE APL,
held in Paris, France, September 9 and 10, 1971;
and appears in the proceedings of that conference.

ABSTRACT

APL primitives are formally defined by APL/360 functions. The description is formal in two senses: primitives are completely and exactly defined for all cases, and the functions are executable on APL/360 and are hence working models.

The descriptions can be used to compare and evaluate APL implementations in two ways:

1. Implemented primitives should produce the same results as the corresponding definitions.

2. Any implementation should properly execute the definitions.

INTRODUCTION

This is a description of the primitives of APL. They are defined by APL\360 functions which describe them to the approximate extent of the implementation of APL\360. The APL\360 User's Manual is the principal reference, and familiarity with it is assumed.

The description is formal in two senses: primitives are defined completely and exactly for all cases; and the functions which form the description are executable and hence are working models. Our intent was to describe the primitives of APL more completely and more rigorously than does the APL\360 User's Manual, but we did not intend the description to be documentation for any specific implementation. Hence, we have tended to ignore machine-dependent and system-design considerations such as library structures and the mechanics of function definition.

This description can be used to compare and evaluate implementations in two ways:

1. With the exception of machine and system dependencies, implemented primitives should behave like the corresponding definitions for the same arguments.

2. Since the functions forming the descriptions are themselves executable APL, any implementation should execute them properly.

APL was chosen as the language for the description because it allows short and concise yet complete and precise definitions. It is deficient primarily in primitives for constructing and manipulating arrays with components more complicated than scalars; this deficiency makes it impossible to formally define and simulate APL indexing.

We have chosen not to include a formal definition of function definition and statement parsing because such a definition adds detail which is really not required for understanding. Such a definition would in fact be a complete APL interpreter.

APL SYNTAX

The formal syntax of APL is relatively simple. Essentially, a program consists of a sequence of statements which can be parsed into simple expressions. Expressions and statements can be informally defined as follows:

C	is a numeric or character constant,
N	is an undefined name or a variable or a ⎕,
V	is a variable,
E	is an expression,
F	is a function,
P	is a primitive function (see note 1),
S	is a primitive scalar dyadic function,
H	is a statement label,
L	is a semi-colon list.

An expression, *E*, has one of the forms:

```
C
H
V
F            niladic

F E          monadic
P[E] E       (see note 1)

E F E        dyadic
E P[E] E     (see note 1)

S/E          reduction
S/[E] E

S\E          scan
S\[E] E

E ∘.S E      outer product

E S.S E      inner product

E[L]         subscripted (see note 2)

N←E          specification
V[L]←E
```

A statement has one of the forms:

```
E     (see note 3)
H:E

→E    branch
H:→E
```

Notes:

1. See Table 1 for those primitives for which a subscript has meaning.

2. The semi-colon list L in $V[L]$ and $E[L]$ is of the form $E;E;$ ... , where the number of semicolons must be $\bar{}1+\rho\rho V$ and $\bar{}1+\rho\rho E$ respectively. Any of the expressions in the list may be elided, implying all permissible values for that subscript position in ascending sequence.

3. When an expression E does not contain a specification or branch as the function of least precedence, it is assumed that ⎕← was elided on the left, and that the value, if any, of the expression is to be displayed.

Order of evaluation. The relative precedence of functions in an expression is positional rather than attributive: precedence increases from left to right. Parentheses may be used to delimit expressions, and the arguments of a function must be evaluated before the function can be evaluated. The rule governing the order of evaluation within a statement is this: The rightmost function whose arguments are available (i.e. have been evaluated or require no evaluation) is evaluated. Thus, for example, the commutativity of + is maintained in expressions of the form

```
(A←A×2)+A, equivalent to A+A←A×2
  2 1  3                  3 2 1 (evaluation order)
```

INFORMAL DEFINITIONS

The arguments and results of APL functions are scalars or arrays of scalars; a scalar is a number or a character. Characters are distinguished from numbers in that no arithmetic functions are defined on them. Most current implementations require that all scalars forming an array be of the same type, but this restriction is not essential. In this paper, the only test which prevents mixing numbers and characters is the test in *COMMACHECK*. However, no mixed numeric and character constants yet exist in APL, and some of the functions here make the (usually trivial) assumption that an argument is of uniform type.

A vector is a sequence of scalars, formed either by writing a constant vector, by catenation, or by an expression involving a vector. The reshape function, ρ, can be used to reshape its right argument into an array which has the dimensions specified by the left argument. The elements of the result are filled in principal order (right-most subscript changing fastest) with scalars chosen in principal order from the right argument. If the right argument is exhausted, it is repeated cyclically.

Examples:

```
      4⍴1 2 3
  1  2  3  1

      2 3⍴1 2 3 4 5 6
 1 2 3
 4 5 6
```

The monadic function ⍴ returns a vector of the dimensions of its argument. If *B* is the result of an expression, then:

⍴*B*	is a vector of dimensions of the axes of *B*,
⍴⍴*B*	is the number of axes of *B*, and
×/⍴*B*	is the number of elements in *B*.

Specification, the APL primitive function denoted by the symbol ← forms an association between the right argument and the left argument (which must be a name). The result of a specification expression is the right argument. If the name to the left of the ← is indexed, then only the elements designated by the indices are affected.

Indexing is a function which selects a subarray from an APL array, and is the only function currently permitted in the left argument of specification. Vector Indexing is the process of selecting particular components of a vector. If *X* is a vector and *I* is a scalar such that *I*∊⍳⍴*X*, then *X*[*I*] selects the scalar element of *X* located by (*I*=⍳⍴*X*)/*X*. If *I* is not a scalar then the elements of *X*[*I*] are obtained by evaluating the scalar for each element of *I*. In all cases, ⍴*X*[*I*] is equal to ⍴*I*.

The index origin is the value of *I* which selects the first component of a vector - in 0-origin indexing the first component of the vector *X* is *X*[0]; in 1-origin indexing, it is *X*[1] If *Q̲* is the value of the index origin, then all indexing can be expressed in origin zero as *X*[*I*-*Q̲*].

In general, the components of any array *A* can be selected by the expression *A*[*L*] where *L* is a semi-colon list containing ⍴⍴*A* list elements (which may be arrays) separated by ¯1+⍴⍴*A* semi-colons. If *L* is of the form *I*;*J*;*K*;..., then ⍴*A*[*L*] is (⍴*I*),(⍴*J*),(⍴*K*),...

If all of the elements of *L* are scalars, and *M* is the vector formed by catenating these scalars (i.e. *M*←*I*,*J*,*K*,...), then, in terms of vector indexing, the indicated component of *A* is obtained by (,*A*)[(⍴*A*)⊥*M*-*Q̲*] (origin zero).

An item of *L* may be elided, i.e., the four sequences [; ;; ;] and [] are all valid. When the *I*th position is vacant, then the value ⍳(⍴*A*)[*I*] meaning a vector of all permissible values in this position in ascending order is assumed. When any of the elements is not a scalar, the result is determined by applying the scalar case to all combinations taking one scalar from each list element.

When an index appears to the left of specification in the form *A*[*L*]←*V*, the conformability requirement is ∧/(⍴*V*)=(⍴*I*),(⍴*J*),(⍴*K*), ...

Function Execution. A defined function consists of a header and a sequence of statements. Statements are numbered sequentially beginning with 1, and the header is referred to as statement 0. Function headers, local variables, and statement labels are described in detail in the APL\360 User's Manual. Functions are executed beginning with statement 1 and, at the completion of any statement, proceeding with the next in succession. This sequence may be altered by a branch statement. A branch statement is a statement formed by the character → followed by an expression. The result of this expression must be a scalar or a vector, and if non-empty, the first element *X* must be an integer. If *X*∊⍳*N* (origin 1), where *N* is the number of statements in the function, then the next statement executed will be statement *X*. Otherwise, the function terminates, and execution resumes at the point where the function was invoked. If the result of the expression to the right of the → is an empty vector, then the sequence of execution is not changed, and execution proceeds with the succeeding statement. Execution of a function terminates after the last statement of a function unless it is a branch.

External Appearance. The external appearance of APL, which is described in detail in the APL\360 User's Manual, will not be treated formally. The notation used in the definitions is APL\360. APL\360 has many important concepts which we have not treated formally, but which nevertheless are important to the utility of an implementation. We have ignored the library organization and system commands. One very important notion is visual fidelity. The appearance of a typewritten line corresponds as closely as possible to the internal representation of that line. Typing errors are corrected in such a way that the statement of correction and the correction are both clear and legible.

Errors are handled in APL\360 in a way which facilitates recovery and interaction. All errors are detected during execution, even though it is possible to detect some errors earlier. When an error occurs, no action other than suspension of execution is taken. The user is then free to examine the situation at the point where the error occurred, and may correct the error and resume execution if he desires. The place and manner in which errors are detected are shown in each function, and the action following detection consists of printing the type of error and then terminating execution. The functions usually show the detection of errors by a sequence of tests rather than by single more complicated expressions. This was done in order to preserve as much information as possible about the errors to ease possible changes in messages. For example, it might be desirable to differentiate *LEFT DOMAIN* and *RIGHT DOMAIN* errors.

Arithmetic and Fuzz. One of the principal underlying assumptions of APL is that the arithmetic primitives are defined on the entire continuous domain of real numbers, and that arithmetic is exact. We have tended to follow this assumption in the formal definitions, so that, even though the notation is APL\360, division, floor, ceiling, and the relationals are assumed to be exact.

The notion of fuzziness has proved to be so useful in hiding the minor errors caused by finite precision and inexact representation, that we have included it in definitions of floor, ceiling, and the relationals. The definitions are such that mathematical identities are preserved. For example, $A<B$ always implies $B\geq A$.

Programming Conventions. Since the primary intent was to communicate the definitions of APL primitives, we have tended to disregard execution efficiency. The functions are meant to be executed, and in order to provide as much information as possible about their execution through the APL\360 tracing facilities, we have minimized the use of multiple specifications and have avoided the use of specification in a branch statement. The use of indexing has been restricted to the indexing of vectors by scalars or vectors.

The primitive functions which are currently considered part of APL are shown in Table 1. All of these have been represented by functions except for: the simple arithmetic functions + - × ÷, monadic and dyadic ⍴, indexing, specification, and branching. The arithmetic functions have their usual mathematical definitions; the others are treated informally below.

Approximating Transcendental Functions. We have given simple series or continued-fraction approximations for the transcendental functions in order to present executable functions. These approximations are not particularly accurate and are not recommended for implementation. In general, approximating functions should be designed and tailored for the specific host hardware.

Loops and Tests. In general, the order in which loops are executed, as well as the order in which tests are made, is unimportant. For example, when a scalar function is extended to the scalar elements of an array, it doesn't matter which elements are chosen first, as long as all elements of the result are correctly calculated and stored. Thus, unless a loop contains an expression which clearly depends on the sequence followed by the induction variable, no inherent meaning should be ascribed to the order. This is also true for some sequences of statements.

	MONADIC		DYADIC	
Sym	Type	Function	Type	Function
+	s	*MPLUS*	sp	–
-	s	*NEGATE*	sp	–
×	s	*SIGNUM*	sp	–
÷	s	*RECIP*	sp	–
⌈	s	*TCL, FCL*	s	*MAX*
⌊	s	*TFL, FFL*	s	*MIN*
*	s	*ETO*	s	*EXP*
⍟	s	*LN*	s	*LOG*
|	s	*ABS*	s	*RES*
!	s	*SHRIEK*	s	*BC*
?	so	*ROLL*	mo	*DEAL*
○	s	*PITIMES*	s	*CIRCLE*
~	s	*NOT*	u	–
∧	u	–	s	*AND*
∨	u	–	s	*OR*
⍲	u	–	s	*NAND*
⍱	u	–	s	*NOR*
<	u	–	s	*FLT*
≤	u	–	s	*FLE*
=	u	–	s	*FEQ*
≥	u	–	s	*FGE*
>	u	–	s	*FGT*
≠	u	–	s	*FNE*
⍴	mp	–	mp	–
,	m	*RAVEL*	mi	*COMMA*
⍳	mo	*XGEN*	mo	*XOF*
↑	u	–	m	*TAKE*
↓	u	–	m	*DROP*
⍋	mio	*GRADEUP*	u	–
⍒	mio	*GRADEDOWN*	u	–
/	u	–	mi	*COMPRESS*
\	u	–	mi	*EXPAND*
⌽	mi	*REVERSE*	mi	*ROTATE*
⍉	m	*MTRANSPOSE*	mo	*TRANSPOSE*
∈	u	–	m	*MEMBER*
⊥	u	–	m	*DECODE*
⊤	u	–	m	*ENCODE*
⌹	m	*MMD*	m	*DMD*
←	u	–	p	–

Types: s - scalar function
m - mixed function
i - index has meaning and is origin dependent
o - function is origin dependent
p - primitive to this report, no formal definition given
u - no definition exists

TABLE 1. APL Primitive Functions

Dyadic Function		Identity Element	Left-Right
Times	×	1	L R
Plus	+	0	L R
Divide	÷	1	R
Minus	−	0	R
Power	*	1	R
Logarithm	⍟		None
Maximum	⌈	–	L R
Minimum	⌊		L R
Residue	|	0	L
Circle	○		None
Out of	!	1	L
Or	∨	0	L R
And	∧	1	L R
Nor	⍱		None
Nand	⍲		None
Equal	=	1 Apply	L R
Not equal	≠	0 for	L R
Greater	>	0 logical	R
Not less	≥	1 arguments	R
Less	<	0 only	L
Not greater	≤	1	L

TABLE 2. Identity Elements of Dyadic Scalar Functions

VARIABLES

Global Variables. The following variables are parameters to the execution of APL expressions, and can be examined and modified by the user.

B - the last random number generated in *ROLL*. An integer such that (*B*≥0)∧*B*<*P*

N - the number of bits ignored in comparisons.

Q - the value of the index origin, a scalar 0 or 1.

The following represent values which are usually determined by hardware:

C - the character set. In APL\360, there are 256 distinct characters. Approximately 150 of these have associated printing graphics.

NB - the floating-point number base.

WL - the number of digits in the floating-point fraction.

P and *Q* are parameters of the random number generator. In APL\360 they are ¯1+2*31 and 7*5 respectively. Generally, *P* is chosen to be the largest prime which can be stored in the machine accumulator and *Q* is a primitive root of *P*.

The remaining global variables are used by the definitions to save and pass information:

$\underline{F}$ - a character scalar which contains the APL symbol corresponding to the scalar dyadic function represented by the example function *F*.

$\underline{F}1$ - the identity element of the scalar dyadic function represented by *F*.

$\underline{G}$ - a character scalar which contains the APL symbol which denotes the scalar dyadic function represented by the example function *G*.

$\underline{I}$ - a scalar integer which is used by *AXIS* to pass a subscript value to an indexable function. 0=ρ$\underline{I}$ is used to indicate an elided index, and therefore each function which uses a function index must set $\underline{I}$←⍳0 before terminating.

Local Variables. The following conventions are used for naming local variables:

Z - the function result.

A - the left argument.

B - the right argument.

I - axis of application.

```
RVLA = ,A
RVLB = ,B
RVLZ = ,Z

CA = ×/A
CB = ×/B
CZ = ×/Z

LA = ρA
LB = ρB
LZ = ρZ

XLA = (ρA)[I]
XLB = (ρB)[I]
XLZ = (ρZ)[I]

TCA = ×/I↓ρA
TCB = ×/I↓ρB
TCZ = ×/I↓ρZ
```

SCALAR FUNCTIONS

The scalar functions are those functions, like +, which are defined on scalars and give a scalar result. These functions are extended to arrays by the adverb functions *IP*, *OP*, *R*, *SCAN*, and *SD*.

```
    ∇ Z←ABS B
[1]   ⍝SCALAR FUNCTION Z←|B
[2]   Z←B⌈-B
    ∇

    ∇ Z←MPLUS B
[1]   ⍝SCALAR FUNCTION Z←+B
[2]   Z←0+B
    ∇

    ∇ Z←NEGATE B
[1]   ⍝SCALAR FUNCTION Z←-B
[2]   Z←0-B
    ∇

    ∇ Z←SIGNUM B
[1]   ⍝SCALAR FUNCTION  Z←×B
[2]   Z←(B>0)-B<0
    ∇

    ∇ Z←RECIP B
[1]   ⍝SCALAR FUNCTION Z←÷B
[2]   Z←1÷B
    ∇

    ∇ Z←A MIN B
[1]   ⍝SCALAR FUNCTION: Z←A⌊B
[2]   Z←A
[3]   →0 IF A≤B
[4]   Z←B
    ∇

    ∇ Z←A MAX B
[1]   ⍝SCALAR FUNCTION Z←A⌈B
[2]   Z←A
[3]   →0 IF A≥B
[4]   Z←B
    ∇

    ∇ Z←A AND B
[1]   ⍝SCALAR FUNCTION Z←A∧B
[2]   'DOMAIN' ERROR~A∊ 0 1
[3]   'DOMAIN' ERROR~B∊ 0 1
[4]   Z←A⌊B
    ∇

    ∇ Z←A OR B
[1]   ⍝SCALAR FUNCTION Z←A∨B
[2]   'DOMAIN' ERROR~A∊ 0 1
[3]   'DOMAIN' ERROR~B∊ 0 1
[4]   Z←A⌈B
    ∇
```

```
    ∇ Z←A NAND B
[1]   ⍝SCALAR FUNCTION Z←A⍲B
[2]   Z←~A∧B
    ∇

    ∇ Z←A NOR B
[1]   ⍝SCALAR FUNCTION Z←A⍱B
[2]   Z←~A∨B
    ∇

    ∇ Z←NOT B
[1]   ⍝SCALAR FUNCTION Z←~B
[2]   'DOMAIN' ERROR~B∊ 0 1
[3]   Z←1-B
    ∇

    ∇ Z←PITIMES B
[1]   ⍝SCALAR FUNCTION Z←○B
[2]   Z←B×3.141592653589793
    ∇

    ∇ Z←SHRIEK B
[1]   ⍝SCALAR FUNCTION Z←!B
[2]   ⍝USES:FACT
[3]   'DOMAIN' ERROR 0≠TYPE B
[4]   'DOMAIN' ERROR(B<0)∧B=⌊B
[5]   Z←FACT B
    ∇

    ∇ Z←ROLL B
[1]   ⍝SCALAR FUNCTION Z←?B
[2]   ⍝GLOBAL VARIABLES:B̲ O̲ P̲ Q̲
[3]   'DOMAIN' ERROR 0≠TYPE B
[4]   'DOMAIN' ERROR(0≥B)∨B≠⌊B
[5]   B̲←P̲|B̲×Q̲
[6]   Z←O̲+⌊B×B̲÷P̲
    ∇

    ∇ Z←A BC B;C;ANI;BNI;CNI
[1]   ⍝SCALAR FUNCTION Z←A!B
[2]   ⍝USES:FACT
[3]   'DOMAIN' ERROR 0≠TYPE A
[4]   'DOMAIN' ERROR 0≠TYPE B
[5]   C←B-A
[6]   ANI←(A<0)∧A=⌊A
[7]   BNI←(B<0)∧B=⌊B
[8]   CNI←(C<0)∧C=⌊C
[9]   'DOMAIN' ERROR BNI∧A≠⌊A
[10]  Z←0
[11]  →0 IF(BNI∧ANI∧CNI)∨(~BNI)∧ANI∨CNI
[12]  →GAMMA IF(A≠⌊A)∨B≠⌊B
[13]  A←(|A)⌊|B-A
[14]  Z←1
[15] L:→0 IF 0=A
[16]  Z←Z×B÷A
[17]  B←B-1
[18]  A←A-1
[19]  →L
[20] GAMMA:Z←(FACT B)÷(FACT A)×FACT C
    ∇
```

```
    ∇ Z←LN B
      ⍝SCALAR FUNCTION Z←⍟B
      'DOMAIN ' ERROR B≤0
      B←B-1
      Z←B÷1+B÷2+B÷3+(4×B)÷4+(4×B)÷5+(9×B)÷6
    ∇
```

```
    ∇ Z←ETO B
      ⍝SCALAR FUNCTION Z←*B
      Z←1÷1-B÷1+B÷2-B÷3+B÷2-B÷5+B÷2
    ∇
```

```
    ∇ Z←A LOG B
      ⍝SCALAR FUNCTION Z←A⍟B
      Z←(⍟B)÷⍟A
    ∇
```

Exponentiation of Negative Numbers. When *A<0* in *A*B*, the function *RAPPROX* is used to determine whether *B* can be approximated by a rational number of the form *P÷Q*. If it cannot be, then the result is -(|*A*)**B*. If it can, and *Q* is even, then *A*B* is not defined. If *Q* is odd and *P* is even, the result is (|*A*)**B*, and if both *P* and *Q* are odd, the result is -(|*A*)**B*. *RAPPROX* returns a 2 2 matrix. The two elements of the last column specify the parity of *P* and *Q* respectively, 0 meaning even and 1 denoting odd. If *B* is irrational, then the result of *RAPPROX* is 2 2⍴¯1.

```
    ∇ Z←A EXP B
      ⍝SCALAR FUNCTION Z←A*B
      ⍝USES:RAPPROX
      →S IF A≥0
      P←, 0 1 ↓RAPPROX B
      'DOMAIN ' ERROR 0=1↓P
      Z←(¯1*1↑P)××B×⍟|A
      →0
     S:Z←*B×⍟A
    ∇
```

```
    ∇ P←RAPPROX X;N;E;B;T
      ⍝USES:FUZZ
      P← 1 2 ∘.= 1 2
      N←10
      E←FUZZ
      B←X
     IT:→0 IF B≤E
      →IR IF 0≥N←N-1
      T←1÷B
      X←⌊T
      P←⌽P
      P[;2]←P[;2]≠P[;1]∧2|X
      B←T-X
      E←FUZZ+E×T×T
      →IT
     IR:P← 2 2 ⍴¯1
    ∇
```

FACT is used by *SHRIEK* and *BC*. It is similar to the gamma function, but differs in that it is defined for negative integers. ! is used on lines 15 and 19 to calculate the gamma function in the domain from 0 to 1.

```
    ∇ Z←FACT B;I;U;F
[1]   Z←1
[2]   F←B-⌊B
[3]   →NEG1 IF B<0
[4]   I←1+F
[5]   U←1+B
[6]   →L
[7]  NEG1:I←1+B
[8]   U←F
[9]  L:→E IF U=I
[10]  Z←Z×I
[11]  I←I+1
[12]  →L
[13] E:→NEG2 IF B<0
[14]  →0 IF 0=F
[15]  Z←Z×!F
[16]  →0
[17] NEG2:Z←÷Z
[18]  →0 IF 0=F
[19]  Z←(Z×!F)÷F
    ∇

    ∇ Z←A RES B;I;C
[1]   ⍝SCALAR FUNCTION Z←A|B
[2]   Z←B
[3]   →0 IF(0=A)∨0=B
[4]   I←A×¯1*(×A)≠×B
[5]   C←|A
[6]  L:→E IF 0<C-|Z
[7]   Z←Z-I
[8]   →L
[9]  E:→0 IF(×A)=×B
[10]  Z←Z+A
    ∇

    ∇ Z←A CIRCLE B
[1]   ⍝SCALAR FUNCTION Z←A○B
[2]   'DOMAIN' ERROR~A∊¯8+⍳15
[3]   →(L̲7,L̲6,L̲5,L̲4,L̲3,L̲2,L̲1,L0,L1,L2,L3,L4,L5,L6,L7)[A+8]
[4]  L̲7:Z←ATANH B
[5]   →0
[6]  L̲6:Z←ACOSH B
[7]   →0
[8]  L̲5:Z←ASINH B
[9]   →0
[10] L̲4:Z←CIRCLE̲4 B
[11]  →0
[12] L̲3:Z←ARCTAN B
[13]  →0
[14] L̲2:Z←ARCCOS B
[15]  →0
[16] L̲1:Z←ARCSIN B
[17]  →0
[18] L0:Z←CIRCLE0 B
[19]  →0
```

```
[20] L1:Z←SIN B
[21]  →0
[22] L2:Z←COS B
[23]  →0
[24] L3:Z←TAN B
[25]  →0
[26] L4:Z←CIRCLE4 B
[27]  →0
[28] L5:Z←SINH B
[29]  →0
[30] L6:Z←COSH B
[31]  →0
[32] L7:Z←TANH B
    ∇


    ∇ Z←ATANH B;B2
[1]    ⍝SCALAR FUNCTION Z←¯7○B
[2]    'DOMAIN' ERROR 1≤|B
[3]    B2←B×B
[4]    Z←B÷1-B2÷3-4×B2÷5-9×B2÷7-16×B2÷9-25×B2÷11-36×B2÷13
    ∇

    ∇ Z←ACOSH B
[1]    ⍝SCALAR FUNCTION Z←¯6○B
[2]    'DOMAIN' ERROR 1>B
[3]    Z←⍟B+¯4○B
    ∇

    ∇ Z←ASINH B;X;X2
[1]    ⍝SCALAR FUNCTION Z←¯5○B
[2]    Z←(⍟(|B)+4○B)×¯1*B<0
    ∇

    ∇ Z←CIRCLE4 B
[1]    ⍝SCALAR FUNCTION Z←¯4○B
[2]    'DOMAIN' ERROR 1>|B
[3]    Z←(¯1+B×B)*0.5
    ∇

    ∇ Z←ARCTAN B;X;X2
[1]    ⍝SCALAR FUNCTION Z←¯3○B
[2]    X←|B
[3]    X2←X×X
[4]    Z←X÷1+X2÷3+4×X2÷5+9×X2÷7+16×X2÷9
[5]    Z←Z×¯1*B<0
    ∇

    ∇ Z←ARCCOS B
[1]    ⍝SCALAR FUNCTION Z←¯2○B
[2]    'DOMAIN' ERROR 1<|B
[3]    Z←(○0.5)-¯1○|B
[4]    →0 IF 0≤B
[5]    Z←(○1)-Z
    ∇
```

```
    ∇ Z←ARCSIN B;X;X2
[1]   ⍝SCALAR FUNCTION Z←¯1○B
[2]   'DOMAIN' ERROR 1<|B
[3]   X2←X×X
[4]   Z←((1-X2)*0.5)×X÷1-2×X2÷3-2×X2÷5-12×X2÷7-12×X2÷9
    ∇

    ∇ Z←CIRCLE0 B
[1]   ⍝SCALAR FUNCTION Z←0○B
[2]   'DOMAIN' ERROR 1<|B
[3]   Z←(1-B×B)*0.5
    ∇

    ∇ Z←SIN B;C
[1]   ⍝SCALAR FUNCTION Z←1○B
[2]   C←1+2×¯1+⍳15
[3]   Z←-/(B*C)÷!C
    ∇

    ∇ Z←COS B
[1]   ⍝SCALAR FUNCTION  Z←2○B
[2]   Z←1○(○0.5)-B
    ∇

    ∇ Z←TAN B
[1]   ⍝SCALAR FUNCTION Z←3○B
[2]   Z←(1○B)÷2○B
    ∇

    ∇ Z←CIRCLE4 B
[1]   ⍝SCALAR FUNCTION  Z←4○B
[2]   Z←(1+B×B)*0.5
    ∇

    ∇ Z←SINH B
[1]   ⍝SCALAR FUNCTION Z←5○B
[2]   Z←0.5×-/*B,-B
    ∇

    ∇ Z←COSH B
[1]   ⍝SCALAR FUNCTION  Z←6○B
[2]   Z←0.5×+/*B,-B
    ∇

    ∇ Z←TANH B
[1]   ⍝SCALAR FUNCTION Z←7○B
[2]   Z←(5○B)÷6○B
    ∇
```

SCALAR FUNCTIONS THAT USE FUZZ

The primary use of fuzz occurs in floor, ceiling, and the relationals. The first two functions below define absolute and relative fuzz, respectively. The three global variables used have the following meanings:

NB The base used to represent the floating-point fraction. In System/360, the base is 16.

WL The number of digits, base *NB*, forming the floating-point fraction. In System/360, the number of digits in the fraction is 14.

N The number of bits, or binary digits, to be ignored in comparisons. As the functions demonstrate, this notion is valid on machines which do not use binary encoding, and Wilkinson error analysis techniques are valid.

```
    ∇ Z←FUZZ
[1]   ⍝GLOBAL VARIABLES:N NB WL
[2]   Z←((2*N)-1)×NB*-WL
    ∇

    ∇ Z←A RFUZZ B
[1]   ⍝GLOBAL VARIABLES:NB
[2]   ⍝USES:FUZZ
[3]   ⍝RELATIVE FUZZ
[4]   Z←FUZZ×NB*⌈NB⍟(|A)⌈|B
    ∇
```

True ceiling and true floor have been included for illustration and emphasis in the other functions.

```
    ∇ Z←TCL B
[1]   ⍝TRUE CEILING
[2]   Z←B+1|-B
    ∇

    ∇ Z←TFL B
[1]   ⍝TRUE FLOOR
[2]   Z←B-1|B
    ∇

    ∇ Z←FCL B
[1]   ⍝SCALAR FUNCTION  Z←⌈B
[2]   ⍝USES:FUZZ FGT TFL FEQ TCL
[3]   Z←FUZZ<|B
[4]   →0 IF~Z
[5]   Z←TCL B
[6]   →0 IF(B FGT TFL B)∨B FEQ Z
[7]   Z←TFL B
    ∇

    ∇ Z←FFL B
[1]   ⍝SCALAR FUNCTION Z←⌊B
[2]   ⍝USES:FUZZ TFL FLT TCL FEQ
[3]   Z←FUZZ<|B
[4]   →0 IF~Z
[5]   Z←TFL B
[6]   →0 IF(B FLT TCL B)∨B FEQ Z
[7]   Z←TCL B
    ∇

    ∇ Z←A FLT B
[1]   ⍝SCALAR FUNCTION Z←A<B
[2]   ⍝USES:RFUZZ
[3]   Z←A<B-A RFUZZ B
    ∇

    ∇ Z←A FLE B
[1]   ⍝SCALAR FUNCTION Z←A≤B
[2]   ⍝USES:RFUZZ
[3]   Z←A≤B+A RFUZZ B
    ∇
```

```
    ∇ Z←A FEQ B
[1]   ⍝SCALAR FUNCTION Z←A=B
[2]   ⍝USES:RFUZZ
[3]   Z←(A RFUZZ B)≥|A-B
    ∇

    ∇ Z←A FGE B
[1]   ⍝SCALAR FUNCTION Z←A≥B
[2]   ⍝USES:RFUZZ
[3]   Z←A≥B-A RFUZZ B
    ∇

    ∇ Z←A FGT B
[1]   ⍝SCALAR FUNCTION Z←A>B
[2]   ⍝USES:RFUZZ
[3]   Z←A>B+A RFUZZ B
    ∇

    ∇ Z←A FNE B
[1]   ⍝SCALAR FUNCTION Z←A≠B
[2]   ⍝USES:RFUZZ
[3]   Z←(A RFUZZ B)<|A-B
    ∇
```

ADVERB FUNCTIONS

Scalar functions can be extended to arrays in four ways: Simple Scalar Extension, Reduction, Outer Products, and Inner Products.

Scalar Extension. This function extends scalar functions to arrays element by element. The scalar function is represented by the function *F* which may be any of the scalar dyadic functions. *SD* performs conformability and general domain checks, but assumes that *F* will detect most domain errors.

```
    ∇ Z←A SD B;CZ;RVLZ;RVLA;RVLB;I
[1]   ⍝GLOBAL VARIABLES:F̲
[2]   ⍝USES:F
[3]   'DOMAIN' ERROR(0≠TYPE A)∧~F̲∊'≠='
[4]   'DOMAIN' ERROR(0≠TYPE B)∧~F̲∊'≠='
[5]   →SINGULAR IF(1=×/⍴A)∨1=×/⍴B
[6]   'RANK' ERROR(⍴⍴A)≠⍴⍴B
[7]   'LENGTH' ERROR∨/(⍴A)≠⍴B
[8]   →L1 IF~0∊⍴A
[9]   Z←A
[10]  →0
[11] SINGULAR:→(ASINGULAR,BSINGULAR) IF((1≠×/⍴B)∨(⍴⍴A)<⍴⍴B),
      (1≠×/⍴A)∨(⍴⍴A)>⍴⍴B
[12]  →L1
[13] ASINGULAR:→L2 IF 0≠×/⍴B
[14]  Z←B
[15]  →0
[16] L2:A←(⍴B)⍴A
[17]  →L1
[18] BSINGULAR:→L3 IF 0≠×/⍴A
[19]  Z←A
[20]  →0
```

```
[21] L3:B←(⍴A)⍴B
[22] L1:CZ←×/⍴A
[23]  RVLZ←CZ⍴0
[24]  RVLA←,A
[25]  RVLB←,B
[26]  I←0
[27] LOOP:I←I+1
[28]  →END IF I>CZ
[29]  RVLZ[I]←RVLA[I] F RVLB[I]
[30]  →LOOP
[31] END:Z←(⍴A)⍴RVLZ
    ∇
```

Reduction. *SREDUCTION* performs vector reduction. *R* extends vector reduction to higher arrays and performs error checking. The scalar dyadic function which is reducing is represented by *F*. *F̲* contains the character symbol which denotes the reducing function, and *F̲1* is the value of the corresponding identity element.

F/[I]X is represented here as *R I AXIS X* with appropriate specifications of *F*, *F̲*, and *F̲1*.

```
    ∇ Z←R B;LZ;CB;XLB;RVLZ;RVLB;TCB;V;E;M;J;K;I
[1]   ⍝GLOBAL VARIABLES: F̲ F̲1
[2]   ⍝USES:OKINDEX SREDUCTION
[3]   I←OKINDEX B
[4]   'DOMAIN' ERROR(0≠TYPE B)∧~F̲∊'≠='
[5]   →L1 IF 0≠⍴⍴B
[6]   B←,B
[7]   I←,1
[8]  L1:LZ←((¯1+I)↑⍴B),I↓⍴B
[9]   XLB←(⍴B)[I]
[10]  →L2 IF 0≠XLB
[11]  'DOMAIN' ERROR F̲∊'⍟⍱⍲○'
[12]  Z←LZ⍴F̲1
[13]  →L3
[14] L2:Z←LZ⍴B
[15] L3:CB←×/⍴B
[16]  →0 IF(0=CB)∨1=XLB
[17]  RVLZ←,Z
[18]  RVLB←,B
[19]  TCB←×/I↓⍴B
[20]  V←TCB×¯1+⍳XLB
[21]  E←TCB×¯1+XLB
[22]  M←J←1
[23] OUTER:K←1
[24] INNER:RVLZ[M]←SREDUCTION RVLB[J+V]
[25]  K←K+1
[26]  J←J+1
[27]  M←M+1
[28]  →INNER IF K≤TCB
[29]  J←J+E
[30]  →OUTER IF J≤CB
[31]  Z←LZ⍴RVLZ
    ∇
```

```
    ∇ Z←SREDUCTION B;J
[1]    ⍝USES:F
[2]    J←(⍳0)⍴⍴B
[3]    Z←B[J]
[4]  L:J←J-1
[5]    →0 IF J=0
[6]    Z←B[J] F Z
[7]    →L
    ∇
```

Outer Product. *F* represents the scalar dyadic function in *A* ∘.*F B*, and *E̲* contains the character symbol which denotes the function.

```
    ∇ Z←A OP B;CA;CB;CZ;RVLZ;RVLA;RVLB;J;M;K
[1]    ⍝GLOBAL VARIABLES:E̲
[2]    ⍝USES:F
[3]    'DOMAIN' ERROR(0≠TYPE A)∧~E̲∊'≠='
[4]    'DOMAIN' ERROR(0≠TYPE B)∧~E̲∊'≠='
[5]    CA←×/⍴A
[6]    CB←×/⍴B
[7]    CZ←CA×CB
[8]    RVLZ←CZ⍴0
[9]    →END IF 0=CZ
[10]   RVLA←,A
[11]   RVLB←,B
[12]   J←M←0
[13] OUTER:J←J+1
[14]   →END IF J>CA
[15]   K←0
[16] INNER:K←K+1
[17]   →INCREMENT IF K>CB
[18]   RVLZ[M+K]←RVLA[J] F RVLB[K]
[19]   →INNER
[20] INCREMENT:M←M+CB
[21]   →OUTER
[22] END:Z←((⍴A),⍴B)⍴RVLZ
    ∇
```

Inner Products. Inner products of the form *A F.G B* and decode expressions *A*⊥*B* are closely related and are combined here. The main function, *BASEPROD*, distinguishes inner product from decode by the value of the global variable *I̲*, which is set in *DECODE* and *IP*. *BASEPROD* in turn uses *R* to reduce or *SDECODE* for ⊥ between conforming vectors. *F* and *G* denote the first and second inner product functions, and *E̲* is the identity element of *F*.

```
    ∇ Z←A IP B
[1]    ⍝GLOBAL VARIABLES:I̲
[2]    ⍝USES:BASEPROD
[3]    I̲←2
[4]    Z←A BASEPROD B
    ∇

    ∇ Z←A DECODE B
[1]    ⍝GLOBAL VARIABLES:I̲
[2]    ⍝USES:BASEPROD
[3]    I̲←1
[4]    Z←A BASEPROD B
    ∇
```

```
    ∇ Z←A SDECODE B;U;I
      U←(ρA)⌈ρB
      A←UρA
      B←UρB
      I←1
      Z←B[1]
     L:I←I+1
      →0 IF U<I
      Z←B[I]+Z×A[I]
      →L
    ∇

    ∇ Z←A BASEPROD B;LLA;FLB;LZ;CZ;RVLZ;VA;TCB;VB;RVLA;RVLB;J;M;K;I
      ⍝GLOBAL VARIABLES:I̲ E̲1 G̲
      ⍝USES:G R SDECODE
      I←I̲
      I̲←⍳0
      →IP1 IF I>1
      'DOMAIN' ERROR 0≠TYPE A
      'DOMAIN' ERROR 0≠TYPE B
      →L1
     IP1:'DOMAIN' ERROR(0≠TYPE A)∧~G̲∊'≠='
      'DOMAIN' ERROR(0≠TYPE B)∧~G̲∊'≠='
     L1:→L2 IF 0≠ρρA
      A←,A
     L2:→L3 IF 0≠ρρB
      B←,B
     L3:LLA←¯1↑ρA
      FLB←1↑ρB
      'LENGTH' ERROR(1≠LLA)∧(1≠FLB)∧LLA≠FLB
      LZ←(¯1↓ρA),1↓ρB
     IP2:RVLZ←(×/LZ)ρE̲1×I>1
     L4:→END IF 0=×/(ρA),ρB
      VA←¯1+⍳LLA
      TCB←×/1↓ρB
      VB←TCB×¯1+⍳FLB
      CA←×/ρA
      RVLA←,A
      RVLB←,B
      J←~M←0
     OUTER:K←1
     INNER:→IP3 IF I>1
      RVLZ[M+K]←RVLA[J+VA] SDECODE RVLB[K+VB]
      →L5
     IP3:RVLZ[M+K]←R RVLA[J+VA] G RVLB[K+VB]
     L5:K←K+1
      →INNER IF K≤TCB
      M←M+TCB
      J←J+LLA
      →OUTER IF J≤CA
     END:Z←LZρRVLZ
    ∇
```

MIXED FUNCTIONS

Mixed functions generally operate on the structure of their arguments rather than on the values of the elements. Some mixed functions can be written with a subscript in expressions of the form *A F*[*I*] *B* and *F*[*I*] *B*. We represent these expressions, using an auxiliary function, as *A F I AXIS B* and *F I AXIS B* respectively. Eliding *I AXIS* is analagous to eliding [*I*].

Catenation. *A*,[*I*]*B* is represented by *A COMMA I AXIS B*

```
    ∇ Z←A COMMA B;L;I
[1]   ⍝GLOBAL VARIABLES:I̲
[2]   ⍝USES:CAT COMMACHECK
[3]   I←A COMMACHECK B
[4]  LAMINATE:→CATENATE IF I=⌊I
[5]   →(ASCALAR,BSCALAR) IF(0=⍴⍴A),0=⍴⍴B
[6]   'RANK' ERROR(⍴⍴A)≠⍴⍴B
[7]   'LENGTH' ERROR∨/(⍴A)≠⍴B
[8]   →BSCALAR
[9]  ASCALAR:L←⍴B
[10]  →L1
[11] BSCALAR:L←⍴A
[12] L1:L←((⌊I)↑L),1,(⌊I)↓L
[13]  I←⌈I
[14]  A←L⍴A
[15]  B←L⍴B
[16] CATENATE:I̲←I
[17]  Z←A CAT B
    ∇
```

```
    ∇ Z←A COMMACHECK B
[1]   ⍝GLOBAL VARIABLES:I̲
[2]   Z←I̲
[3]   I̲←⍳0
[4]   'DOMAIN' ERROR(TYPE A)≠TYPE B
[5]   'INDEX' ERROR 0≠TYPE Z
[6]   'INDEX' ERROR 2≤⍴⍴Z
[7]   'INDEX' ERROR 1<×/⍴Z
[8]   →L2 IF 0≠×/⍴Z
[9]   Z←0.5
[10]  →0 IF(0=⍴⍴A)∧0=⍴⍴B
[11]  Z←(⍴⍴A)⌈⍴⍴B
[12]  →0
[13] L2:'INDEX' ERROR Z≤0
[14]  'INDEX' ERROR Z≥1+(⍴⍴A)⌈⍴⍴B
    ∇
```

```
    ∇ Z←A CAT B;R;LZ;NOTI;LA;LB;RD;WA;WB;CZ;RVLZ;RVLA;RVLB;TCZ;VA;
      VB;J;K;M;I
[1]   ⍝GLOBAL VARIABLES:I̲
[2]   I←I̲
[3]   I̲←⍳0
[4]   R←(⍴⍴A)⌈⍴⍴B
[5]   LZ←R⍴0
[6]   NOTI←((I-1)↑⍳R),I↓⍳R
[7]   LA←⍴A
[8]   LB←⍴B
```

```
[9]   →(ASCALAR,BSCALAR) IF(0=ρρA),0=ρρB
[10]  RD←(ρρA)-ρρB
[11]  'RANK' ERROR 1<|RD
[12]  →(RD1,RD̲1) IF(RD=1),RD=¯1
[13]  'LENGTH' ERROR∨/(LA≠LB)[NOTI]
[14]  LZ[NOTI]←LA[NOTI]
[15]  WA←LA[I]
[16]  WB←LB[I]
[17]  →L1
[18] RD1:'LENGTH' ERROR∨/LA[NOTI]≠LB
[19]  LZ[NOTI]←LB
[20]  WA←LA[I]
[21]  WB←1
[22]  →L1
[23] RD̲1:'LENGTH' ERROR∨/LA≠LB[NOTI]
[24]  LZ[NOTI]←LA
[25]  WA←1
[26]  WB←LB[I]
[27]  →L1
[28] ASCALAR:LZ[NOTI]←LA←LB[NOTI]
[29]  WA←1
[30]  WB←LB[I]
[31]  →L1
[32] BSCALAR:LZ[NOTI]←LB←LA[NOTI]
[33]  WA←LA[I]
[34]  WB←1
[35] L1:LZ[I]←WA+WB
[36]  →L2 IF 0≠WA
[37]  Z←LZρB
[38]  →0
[39] L2:→L3 IF 0≠WB
[40]  Z←LZρA
[41]  →0
[42] L3:CZ←×/LZ
[43]  →L4 IF 0≠CZ
[44]  Z←LZρTYPE A
[45]  →0
[46] L4:RVLZ←CZρTYPE A
[47]  RVLA←(×/LA)ρA
[48]  RVLB←(×/LB)ρB
[49]  TCZ←×/I↓LZ
[50]  WA←WA×TCZ
[51]  WB←WB×TCZ
[52]  VA←¯1+⍳WA
[53]  VB←¯1+⍳WB
[54]  J←K←M←1
[55] LOOP:RVLZ[M+VA]←RVLA[J+VA]
[56]  M←M+WA
[57]  J←J+WA
[58]  RVLZ[M+VB]←RVLB[K+VB]
[59]  M←M+WB
[60]  K←K+WB
[61]  →LOOP IF M≤CZ
[62]  Z←LZρRVLZ
    ∇
```

Compression. *A/[I]B* is represented by *A COMPRESS I AXIS B*

```
    ∇ Z←A COMPRESS B;XLB;LZ;TCB;V;CZ;RVLZ;RVLB;J;M;K;I
[1]    ⍝USES:OKINDEX
[2]    I←OKINDEX B
[3]    →L1 IF 0≠⍴⍴B
[4]    B←(I←,1)⍴B
[5]   L1:→TEST IF 1≠×/⍴A
[6]    A←(⍴B)[I]⍴A
[7]    →L2
[8]   TEST:'RANK' ERROR 1≠⍴⍴A
[9]    'LENGTH' ERROR(⍴B)[I]≠⍴A
[10] L2:'DOMAIN' ERROR~∧/A∊ 0 1
[11]  LZ←⍴B
[12]  LZ[I]←+/A
[13]  →L3 IF(0≠+/A)∧((⍴A)≠+/A)∧0≠×/⍴B
[14]  Z←LZ⍴B
[15]  →0
[16] L3:TCB←×/I↓⍴B
[17]  V←¯1+⍳TCB
[18]  CZ←×/LZ
[19]  RVLZ←CZ⍴TYPE B
[20]  XLB←(⍴B)[I]
[21]  RVLB←,B
[22]  J←M←1
[23] OUTER:K←1
[24] INNER:→SKIP IF~A[K]
[25]  RVLZ[M+V]←RVLB[J+V]
[26]  M←M+TCB
[27] SKIP:K←K+1
[28]  J←J+TCB
[29]  →INNER IF K≤XLB
[30]  →OUTER IF M≤CZ
[31]  Z←LZ⍴RVLZ
    ∇
```

Expansion. *A\[I]B* is represented by *A EXPAND I AXIS B*

```
    ∇ Z←A EXPAND B;LZ;TCB;V;CZ;RVLZ;RVLB;LA;J;M;K;I
[1]    ⍝USES:OKINDEX
[2]    I←OKINDEX B
[3]    →L1 IF 0≠⍴⍴B
[4]    B←(I←,1)⍴B
[5]   L1:→TEST IF 1≠×/⍴A
[6]    A←,A
[7]    →L2
[8]   TEST:'RANK' ERROR 1≠⍴⍴A
[9]   L2:'DOMAIN' ERROR~∧/A∊ 0 1
[10]   'LENGTH' ERROR(⍴B)[I]≠+/A
[11]  LZ←⍴B
[12]  LZ[I]←⍴A
[13]  →L3 IF(⍴A)≠+/A
[14]  Z←B
[15]  →0
[16] L3:→L4 IF(0≠+/A)∧0≠×/⍴B
[17]  Z←LZ⍴TYPE B
[18]  →0
```

```
[19] L4:TCB←×/I↓ρB
[20]  V←¯1+ιTCB
[21]  CZ←×/LZ
[22]  RVLZ←CZρTYPE B
[23]  LA←ρA
[24]  RVLB←,B
[25]  J←M←1
[26] OUTER:K←1
[27] INNER:→SKIP IF~A[K]
[28]  RVLZ[M+V]←RVLB[J+V]
[29]  J←J+TCB
[30] SKIP:K←K+1
[31]  M←M+TCB
[32]  →INNER IF K≤LA
[33]  →OUTER IF M≤CZ
[34]  Z←LZρRVLZ
    ∇
```

Deal. *A?B* is represented by *A DEAL B*

Two algorithms are used. The first, lines 13 through 19, requires *A* iterations and *B* words of storage. The second, lines 20 through 25, requires at least *A* iterations but only *A* words of storage. The decision on line 10 reflects the relative costs of the two algorithms in APL\360.

```
    ∇ Z←A DEAL B;I;J
[1]   ⍝GLOBAL VARIABLES:Q
[2]   ⍝USES:ROLL
[3]   'RANK' ERROR 1≠×/ρA
[4]   'RANK' ERROR 1≠×/ρB
[5]   'DOMAIN' ERROR 0≠TYPE A
[6]   'DOMAIN' ERROR(A<0)∨A≠⌊A
[7]   'DOMAIN' ERROR 0≠TYPE B
[8]   'DOMAIN' ERROR B≠⌊B
[9]   'DOMAIN' ERROR A>B
[10]  →SHORT IF A<⌊B÷16
[11]  Z←(Q-1)+ιB
[12]  →END IF A=0
[13]  I←0
[14] LOOP:J←1+I+(ROLL B-I)-Q
[15]  I←I+1
[16]  Z[I,J]←Z[J,I]
[17]  →LOOP IF A>I
[18] END:Z←A↑Z
[19]  →0
[20] SHORT:Z←ι0
[21] OUTER:→0 IF A=ρZ
[22] INNER:I←ROLL B
[23]  →INNER IF I∊R
[24]  Z←Z,I
[25]  →OUTER
    ∇
```

Matrix Division. *⌹B* and *A⌹B* are represented by *MMD B* and *A DMD B* respectively.

If *B* is a non-singular matrix, then *Z←A⌹B* is such that *A=B+.×Z*. If *B* is over-specified, then *Z* is a least squares solution. *⌹B* is the matrix inverse of *B*. The function is more completely described in "The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL", M.A. Jenkins, Philadelphia Scientific Center technical report number 320-2989.

```
    ∇ Z←MMD B
      Z←((⍳1↑⍴B)∘.=⍳1↑⍴B)⌹B
    ∇

    ∇ Z←A DMD B;P;LA2;LB2;F;I;J;M2;I2;M1;I1;SIGMA;ALFA;U
      'DOMAIN' ERROR 0≠TYPE A
      'DOMAIN' ERROR 0≠TYPE B
      'RANK' ERROR 2≠⍴⍴B
      'RANK' ERROR~(⍴⍴A)∊ 1 2
      'LENGTH' ERROR(1↑⍴A)≠1↑⍴B
      'LENGTH' ERROR(1↑⍴B)<1↓⍴B
      LA2←,1
      →ON IF 1=⍴⍴A
      LA2←1↓⍴A
     ON:LB2←1↓⍴B
      →AHEAD IF(0≠LA2)∧0≠LB2
      Z←(LB2,LA2)⍴0
      →FIN
     AHEAD:P←⍳1↑⍴B
      F←÷⌈/[1]|B÷⍉(⌽⍴B)⍴⌈/|B
      B←B×(⍴B)⍴F
      B←B,A
      I←0
     LOOP:J←I
      I←I+1
      →END IF LB2<I
      M2←⌈/[1]|(0,-LA2)↓(J,J)↓B
      'DOMAIN' ERROR FUZZ≥⌈/M2
      I2←J+M2⍳⌈/M2
      P[I,I2]←P[I2,I]
      B[;I,I2]←B[;I2,I]
      M1←|J↓B[;I]
      I1←J+M1⍳⌈/M1
      B[I,I1;]←B[I1,I;]
      SIGMA←+/(J↓B[;I])*2
      ALFA←(¯1*0≤B[I;I])×SIGMA*0.5
      U←B[I;I]-ALFA
      B[J↓⍳1↑⍴B;I↓⍳1↓⍴B]←((J,I)↓B)-(U,I↓B[;I])∘.×(÷SIGMA-B[I;I]×
      ALFA)×(U,I↓B[;I])+.×(J,I)↓B
      B[I;I]←ALFA
      →LOOP
     END:Z←(LB2,LA2)⍴0
      I←(⍳0)⍴1+LB2
     QBACK:I←I-1
      →RE IF 0=I
      Z[I;]←((LB2↓B[I;])-(LB2↑B[I;])+.×Z)÷B[I;I]
      →QBACK
     RE:Z←Z[⍋P;]×⍉(⌽⍴Z)⍴F
     FIN:→0 IF 1≠⍴⍴A
      Z←,Z
    ∇
```

Take and Drop. *A↑B* and *A↓B* are represented by *A TAKE B* and *A DROP B* respectively.

```
    ∇ Z←A TAKE B
[1]    ⍝USES:TAKECHECK TAKER
[2]    TAKECHECK
[3]    Z←A TAKER B
    ∇

    ∇ Z←A DROP B
[1]    ⍝ USES: TAKER TAKECHECK
[2]    TAKECHECK
[3]    A←(¯1*0<A)×0⌈(⍴B)-|A
[4]    Z←A TAKER B
    ∇

    ∇ TAKECHECK
[1]    ⍝GLOBAL VARIABLES:A B
[2]    'DOMAIN' ERROR 0≠TYPE A
[3]    'RANK' ERROR 2≤⍴⍴A
[4]    A←,A
[5]    'DOMAIN' ERROR∨/A≠⌊A
[6]    →L1 IF 0=⍴⍴B
[7]    'LENGTH' ERROR(⍴⍴B)≠⍴A
[8]    →0
[9]   L1:B←((⍴A)⍴1)⍴B
    ∇

    ∇ Z←A TAKER B;LZ;LB;QB;QZ;L;C;BI;ZI;J;SB;SZ;SL;P;RVLB;RVLZ
[1]    A←,A
[2]    LZ←|A
[3]    LB←⍴B
[4]    →L1 IF∨/LZ≠LB
[5]    Z←B
[6]    →0
[7]   L1:Z←LZ⍴TYPE B
[8]    →0 IF(0=×/LZ)∨0=×/LB
[9]    QB←(A<0)×0⌈LB-LZ
[10]   QZ←(A<0)×0⌈LZ-LB
[11]   L←LZ⌊LB
[12]   C←×/L
[13]   BI←ZI←C⍴1
[14]   J←⍴⍴B
[15]   SB←SZ←SL←1
[16]  LOOP:P←L[J]|⌊(¯1+⍳C)÷SL
[17]   BI←BI+SB×QB[J]+P
[18]   ZI←ZI+SZ×QZ[J]+P
[19]   SB←SB×LB[J]
[20]   SZ←SZ×LZ[J]
[21]   SL←SL×L[J]
[22]   J←J-1
[23]   →LOOP IF J>0
[24]   RVLZ←,Z
[25]   RVLB←,B
[26]   RVLZ[ZI]←RVLB[BI]
[27]   Z←LZ⍴RVLZ
    ∇
```

Encode. *A⊤B* is represented by *A ENCODE B*.

```
    ∇ Z←A ENCODE B;LZ;CA;CB;E;VA;VZ;RVLZ;RVLA;RVLB;J;M;K
     ⍝USES:SENCODE
     'DOMAIN' ERROR 0≠TYPE A
     'DOMAIN' ERROR 0≠TYPE B
     LZ←(ρA),ρB
     CZ←×/LZ
     CB←×/ρB
     →L1 IF 0≠CZ
     Z←LZρ0
     →0
    L1:→L2 IF 0≠ρρA
     A←,A
    L2:E←×/1↓ρA
     VA←E×¯1+⍳1↑ρA
     VZ←CB×VA
     RVLZ←CZρ0
     RVLA←,A
     RVLB←,B
     J←~M←0
    OUTER:K←1
    INNER:RVLZ[M+K+VZ]←RVLA[J+VA] SENCODE RVLB[K]
     K←K+1
     →INNER IF K≤CB
     M←M+CB
     J←J+1
     →OUTER IF J≤E
     Z←LZρRVLZ
     →0
    ∇
```

```
    ∇ Z←A SENCODE B;I
     Z←(I←ρA)ρ0
    L:→REM IF A[I]=0
     Z[I]←A[I]|B
     →0 IF I=1
     B←(B-Z[I])÷A[I]
     →0 IF 0=B
     I←I-1
     →L
    REM:Z[I]←B
    ∇
```

Membership and Inverse Index. *A∊B* and *A⍳B* are represented by *A MEMBER B* and *A XOF B* respectively. The function *FEQ* is used on line 15 of *XOF* to indicate a fuzzed comparison.

```
    ∇ Z←A MEMBER B
     ⍝GLOBAL VARIABLES:Q̲
     Z←((ρ,B)+Q̲-1)≥(,B)⍳A
    ∇
```

```
    ∇ Z←A XOF B;LA;CB;RVLZ;RVLB;J;K
[1]    ⍝GLOBAL VARIABLES:Q
[2]    ⍝USES:FEQ
[3]    'RANK' ERROR 1≠⍴⍴A
[4]    →L1 IF(0≠⍴A)∧0≠×/⍴B
[5]    Z←(⍴B)⍴Q
[6]    →0
[7]   L1:LA←⍴A
[8]    CB←×/⍴B
[9]    RVLZ←CB⍴Q
[10]   RVLB←,B
[11]   J←0
[12]  OUTER:J←J+1
[13]   →END IF J>CB
[14]   K←0
[15]  INNER:K←K+1
[16]   →EXTRA IF K>LA
[17]   →INNER IF~A[K] FEQ RVLB[J]
[18]  EXTRA:RVLZ[J]←K+Q-1
[19]   →OUTER
[20]  END:Z←(⍴B)⍴RVLZ
    ∇
```

Index Generation. ⍳B is represented by *XGEN B*.

```
    ∇ Z←XGEN B;J
[1]    ⍝GLOBAL VARIABLES:Q
[2]    'DOMAIN' ERROR 0≠TYPE B
[3]    'RANK' ERROR 1≠×/⍴B
[4]    'DOMAIN' ERROR B≠⌊|B
[5]    Z←B⍴0
[6]    J←0
[7]   L:J←J+1
[8]    →0 IF J>B
[9]    Z[J]←J+Q-1
[10]   →L
    ∇
```

Transpose. ⍉B and A⍉B are represented by *MTRANSPOSE B* and *A TRANSPOSE B* respectively.

```
    ∇ Z←MTRANSPOSE B;LB;U;J;LZ;W;CZ;T;RVLZ;RVLB;RZ;S;I;K
[1]    →L1 IF 1<×/⍴B
[2]    Z←B
[3]    →0
[4]   L1:LB←⍴B
[5]    U←1+J←⍴⍴B
[6]    LZ←W←J⍴CZ←1
[7]   LOOP:LZ[U-J]←LB[J]
[8]    T←CZ×LB[J]
[9]    W[U-J]←T+CZ
[10]   CZ←T
[11]   J←J-1
[12]   →LOOP IF J>0
[13]   W←W-CZ
[14]   RVLZ←CZ⍴TYPE B
[15]   RVLB←,B
[16]   RZ←⍴⍴B
[17]   S←RZ⍴I←J←1
```

```
[18] MAINLOOP:K←RZ
[19]  RVLZ[J]←RVLB[I]
[20] SEEK:S[K]←1+S[K]
[21]  →BACKUP IF LZ[K]<S[K]
[22]  I←I+W[K]
[23]  J←J+1
[24]  →MAINLOOP
[25] BACKUP:S[K]←1
[26]  K←K-1
[27]  →SEEK IF 0<K
[28]  Z←LZ⍴RVLZ
    ∇

    ∇ Z←A TRANSPOSE B;LB;RB;RZ;J;W1;W2;LZ;M;CZ;CL;I;BI
[1]   ⍝GLOBAL VARIABLES:Q
[2]   'DOMAIN' ERROR 0≠TYPE A
[3]   'RANK' ERROR 2≤⍴⍴A
[4]   A←,A
[5]   'LENGTH' ERROR(⍴⍴B)≠⍴A
[6]   →L1 IF∨/A≠⍳⍴⍴B
[7]   Z←B
[8]   →0
[9]  L1:'DOMAIN' ERROR∨/(A≠⌊A)∨A<Q
[10]  A←A+1-Q
[11]  RZ←⌈/A
[12]  'DOMAIN' ERROR∨/~(⍳RZ)∊A
[13]  LB←⍴B
[14]  RB←⍴⍴B
[15]  W1←(J←RB)⍴1
[16] LOOP1:W1[J-1]←W1[J]×LB[J]
[17]  J←J-1
[18]  →LOOP1 IF 1<J
[19]  LZ←W2←RZ⍴~J←1
[20] LOOP2:M←J=A
[21]  LZ[J]←⌊/M/LB
[22]  W2[J]←+/M/W1
[23]  J←J+1
[24]  →LOOP2 IF J≤RZ
[25]  CZ←×/LZ
[26]  →L2 IF 0≠CZ
[27]  Z←LZ⍴TYPE B
[28]  →0
[29] L2:CL←¯1+⍳CZ
[30]  I←RZ
[31]  BI←CZ⍴0
[32] LOOP:BI←BI+W2[I]×LZ[I]|CL
[33]  CL←⌊CL÷LZ[I]
[34]  I←I-1
[35]  →LOOP IF 0<I
[36]  Z←LZ⍴(,B)[1+BI]
    ∇
```

Ravel. *,B* is represented by *RAVEL B*.

```
    ∇ Z←RAVEL B
[1]   Z←(×/⍴B)⍴B
    ∇
```

Rotate. *A⌽[I]B* is represented by *A ROTATE I AXIS B*.

```
    ∇ Z←A ROTATE B;CB;XLB;NXLB;TCB;V;E;RVLZ;RVLB;RVLA;J;M;K;I
[1]   ⍝USES:SROTATE OKINDEX
[2]   I←OKINDEX B
[3]   'DOMAIN' ERROR 0≠TYPE A
[4]   'DOMAIN' ERROR 1∊A≠⌊A
[5]   CB←×/⍴B
[6]   XLB←(⍴B)[I]
[7]   NXLB←((¯1+I)↑⍴B),I↓⍴B
[8]   →TEST IF 1≠×/⍴A
[9]   A←NXLB⍴A
[10]  →L1
[11] TEST:'RANK' ERROR(⍴NXLB)≠⍴⍴A
[12]  'LENGTH' ERROR∨/NXLB≠⍴A
[13] L1:→L2 IF(1∊0≠XLB|A)∧(1≠×/XLB)∧0≠CB
[14]  Z←B
[15]  →0
[16] L2:TCB←×/I↓⍴B
[17]  V←TCB×¯1+⍳XLB
[18]  E←TCB×¯1+XLB
[19]  RVLZ←CB⍴TYPE B
[20]  RVLB←,B
[21]  RVLA←,A
[22]  J←M←1
[23] OUTER:K←1
[24] INNER:RVLZ[J+V]←RVLA[M] SROTATE RVLB[J+V]
[25]  K←K+1
[26]  J←J+1
[27]  M←M+1
[28]  →INNER IF K≤TCB
[29]  J←J+E
[30]  →OUTER IF J≤CB
[31]  Z←(⍴B)⍴RVLZ
    ∇

    ∇ Z←A SROTATE B;D;M
[1]   D←⍴B
[2]   M←D|A
[3]   Z←B[(M+⍳D-M),⍳M]
    ∇
```

Index check. The following function is used by mixed indexable functions to check the validity of an index and to supply an assumed value if the index is elided.

```
    ∇ Z←OKINDEX B
[1]   ⍝GLOBAL VARIABLES:I̲
[2]   Z←I̲
[3]   I̲←⍳0
[4]   'INDEX' ERROR 0≠TYPE Z
[5]   'INDEX' ERROR 2≤⍴⍴Z
[6]   Z←,Z
[7]   →L IF 0<⍴Z
[8]   →0 IF 0=⍴⍴B
[9]   Z←⍴⍴B
[10]  →0
[11] L:'INDEX' ERROR 1<⍴Z
[12]  'INDEX' ERROR~Z∊⍳⍴⍴B
    ∇
```

MONADIC INDEXABLE MIXED FUNCTIONS

The monadic functions ⍋ ⍒ ⌽ and scan are all similar in that they are indexed and they preserve the structure of the argument. *MINDEXED* is the common control function.

```
    ∇ Z←A MINDEXED B;CB;XLB;RVLZ;RVLB;TCB;V;E;J;K;I
[1]    ⍝USES:SREVERSE GRADE SSCAN OKINDEX
[2]    I←OKINDEX B
[3]    Z←B
[4]    CB←×/⍴B
[5]    →0 IF 0=CB
[6]    →L1 IF A∊ 1 4
[7]    Z←(⍴B)⍴1
[8]   L1:XLB←(⍴B)[I]
[9]    ⍝FOR SCALAR B XLB=⍳0
[10]   →0 IF 1=×/XLB
[11]   RVLZ←RVLB←,B
[12]   TCB←×/I↓⍴B
[13]   V←TCB×¯1+⍳XLB
[14]   E←TCB×¯1+XLB
[15]   J←1
[16]  OUTER:K←1
[17]  INNER:→(REVERSE,GRADEUP,GRADEDOWN,SCAN) IF A=⍳4
[18]  REVERSE:RVLZ[J+V]←SREVERSE RVLB[J+V]
[19]   →L2
[20]  GRADEUP:RVLZ[J+V]←1 GRADE RVLB[J+V]
[21]   →L2
[22]  GRADEDOWN:RVLZ[J+V]←2 GRADE RVLB[J+V]
[23]   →L2
[24]  SCAN:RVLZ[J+V]←SSCAN RVLB[J+V]
[25]  L2:K←K+1
[26]   J←J+1
[27]   →INNER IF K≤TCB
[28]   J←J+E
[29]   →OUTER IF J≤CB
[30]   Z←(⍴B)⍴RVLZ
    ∇
```

Gradeup and Gradedown. ⍋[*I*]*B* and ⍒[*I*]*B* are represented by *GRADEUP I AXIS B* and *GRADEDOWN I AXIS B* respectively. *GRADE* is called by *MINDEXED* to perform the appropriate vector grade.

```
    ∇ Z←GRADEUP B
[1]    ⍝USES:MINDEXED
[2]    'DOMAIN' ERROR 0≠TYPE B
[3]    Z←2 MINDEXED B
    ∇
```

```
    ∇ Z←GRADEDOWN B
[1]    ⍝USES:MINDEXED
[2]    'DOMAIN' ERROR 0≠TYPE B
[3]    Z←3 MINDEXED B
    ∇
```

```
    ∇ Z←A GRADE B;U;J;K;C
[1]    ⍝ A=1 FOR ⍋, A=2 FOR ⍒
[2]    ⍝GLOBAL VARIABLES:Q̲
[3]    U←⍴B
[4]    Z←J←1
[5]   OUTER:J←J+1
[6]    →END IF U<J
[7]    C←B[J]
[8]    K←J
[9]   INNER:K←K-1
[10]   →EX IF 0=K
[11]   →INNER IF((B[K]>C),B[K]<C)[A]
[12]  EX:Z←(K↑Z),J,K↓Z
[13]   C←B[K+1]
[14]   B[K+1]←B[J]
[15]   B[J]←C
[16]   →OUTER
[17]  END:Z←(Q̲-1)+Z
    ∇
```

Reversal. ⌽[*I*]*B* is represented by *REVERSE I AXIS B*. *SREVERSE* is called by *MINDEXED* to perform a vector reversal.

```
    ∇ Z←REVERSE B
[1]    ⍝USES:MINDEXED
[2]    Z←1 MINDEXED B
    ∇
```

```
    ∇ Z←SREVERSE B
[1]    Z←B[(1+⍴B)-⍳⍴B]
    ∇
```

Scan. *F*\[*I*]*B* is represented by *SCAN I AXIS B*. *F*, called from *SSCAN*, represents a dyadic scalar function. *F̲* contains the character symbol which denotes *F*. *SSCAN* is called by *MINDEXED* to perform a vector scan.

```
    ∇ Z←SCAN B
[1]    ⍝GLOBAL VARIABLES:F̲
[2]    ⍝USES:MINDEXED
[3]    'DOMAIN' ERROR(0≠TYPE B)∧~F̲∊'≠='
[4]    Z←4 MINDEXED B
    ∇
```

```
    ∇ Z←SSCAN B;J;K
[1]    ⍝USES:F
[2]    J←⍴B
[3]    Z←B
[4]   OUTER:K←J
[5]   INNER:K←K-1
[6]    →L IF K=0
[7]    Z[J]←Z[K] F Z[J]
[8]    →INNER
[9]   L:J←J-1
[10]  →OUTER IF J>1
    ∇
```

AUXILIARY FUNCTIONS

AXIS is used to represent function subscripting.

TYPE returns a space if its argument contains characters, otherwise it returns zero.

IF and *ERROR* are used for convenience. The third line of *ERROR* (not strictly APL) in APL\360 causes a return to the last level of immediate execution.

F and *G* are used to represent scalar dyadic functions in reduction, scan, and inner and outer products.

```
    ∇ Z←A AXIS B
[1]   ⍝GLOBAL VARIABLES:Q̲ I̲
[2]   'INDEX' ERROR 0≠TYPE A
[3]   I̲←A+1-Q̲
[4]   Z←B
    ∇

    ∇ Z←TYPE B
[1]   ⍝GLOBAL VARIABLES:C̲
[2]   Z←' '
[3]   →0 IF∨/,B∊C̲
[4]   Z←0
    ∇

    ∇ Z←A IF B
[1]   Z←B/A
    ∇

    ∇ A ERROR B
[1]   →0 IF~∨/B
[2]   A,'ERROR'
[3]   →
    ∇

    ∇ Z←A F B
[1]   Z←A+B
    ∇

    ∇ Z←A G B
[1]   Z←A×B
    ∇
```

INDEX OF FUNCTION DEFINITIONS

ACKNOWLEDGEMENTS

Many people contributed functions and advice. In particular, H. J. Smith Jr. carefully studied the functions and suggested several useful revisions. We are grateful to those people, both in IBM and elsewhere, who read the draft and passed along their comments and suggestions.

BIBLIOGRAPHY

1. Falkoff, A. D., "Formal Description of Processes - The First Step in Design Automation," Proceedings of the SHARE Design Automation Workshop, June, 1965.

2. Falkoff, A. D., "Criteria for a System Design Language," Report on NATO Science Committee Conference on Software Engineering Techniques, 1970.

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

4. Falkoff, A. D., K. E. Iverson, and E. H. Sussenguth, "A Formal Description of System/360," IBM Systems Journal, Vol.3, No.3, 1964.

5. Hutchinson, D. W., "A New Uniform Pseudo-Random Number Generator," CACM 9, 432-433, 1966.

6. Iverson, Kenneth E., A Programming Language, John Wiley and Sons, 1962.

7. Iverson, Kenneth E., "Formalism in Programming Languages," CACM 7, 80-88, 1964.

TECHNICAL REPORT INDEXING INFORMATION

1. AUTHOR(S): Lathwell, R. H. and J. E. Mezei	9. INDEX TERMS FOR THE IBM SUBJECT INDEX: APL Formal Description 21 - Programming
2. TITLE: A Formal Description of APL	
3. ORIGINATING DEPARTMENT: Philadelphia Scientific Center	
4. REPORT NUMBER: 320-3008	
5a. NO. OF PAGES: 35	5b. NO. OF REFERENCES: 7

6a. DATE COMPLETED	6b. DATE OF INITIAL PRINTING	6c. DATE OF LAST PRINTING
November 1971	November 1971	

7. ABSTRACT:

APL primitives are formally defined by APL/360 functions. The description is formal in two senses: primitives are completely and exactly defined for all cases, and the functions are executable on APL/360 and are hence working models.

The descriptions can be used to compare and evaluate APL implementations in two ways:

1. Implemented primitives should produce the same results as the corresponding definitions.

2. Any implementation should properly execute the definitions.

8. DISTRIBUTION LIMITATIONS:

PUBLICATIONS

(previously appeared as Technical Reports)
(as of May 1971)

BALINSKI, M. & K. SPIELBERG, "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative," *Progress in Operations Research, V.III; Relationship between Operations Research and the Computer*, (edited by J. Arnofsky), John Wiley & Sons, 195-292, 1969 [320-2938]

BARD, Y., "On a Numerical Instability of Davidon-Like Methods," *Math. of Computation*, V. 22, No.103, 665-666, July 1968 [320-2913]

BARD, Y. & J. GREENSTADT, "A Modified Newton Method for Optimization with Equality Constraints," *OPTIMIZATION - Symp. of the Inst. of Mathematics and Its Applications, Univ. of Keele, England, 1968* (edited by R. Fletcher , A.E.R.E. Harwell), Academic Press, 299-307, 1969 [320-2948]

BARD, Y., "Proof that $H_k \to G^{-1}$ for the E_I Correction," (an Appendix to "Variations on Variable-Metric Methods" by J. Greenstadt), *Math. of Computation*, V. 24, No. 109, 19-22, January 1970 [320-2901]

BARD, Y., "Comparison of Gradient Methods for the Solution of Non-Linear Parameter Estimation Problems," *SIAM J. of Numer. Anal.*, V.7, 157-186, 1970 [320-2955]

BARD, Y., & L. LAPIDUS, "Nonlinear System Identification," *I & EC Fundamentals*, V.9, 628-633, 1970 [320-2980]

BALAS, *E.*, "Project Scheduling with Resource Constraints," *Applications of Mathematical Programming Techniques* (edited by E.M.L. Beale), English Universities Press, 187-200, 1970 [320-2960]

BALAS, *E.*, "Machine Sequencing: Disjunctive Graphs and Degree-Constrained Subgraphs," *Naval Research Logistics Quarterly*, V.17, 1-9, 1970 [320-2971]

COLVILLE, A.R., "A Comparative Study of Nonlinear Programming Codes," *Proc. of the Princeton Symposium on Mathematical Programming* (edited by H.W. Kuhn), Princeton University Press, 487-502, 1970 [320-2949]

COMBA, P.G., "A Language for Three-Dimensional Geometry," *IBM Syst. J.*, V.7, Nos.3 & 4, 292-306, 1969 [320-2923]

COMBA, P.G., "A Procedure for Detecting Intersections of Three-Dimensional Objects," *J. of ACM*, V.15, No. 3, 354-366, July 1968 [320-2924]

EISENPRESS, H. & J. GREENSTADT, "The Estimation of Nonlinear Econometric Systems," *Econometrica*, V.34, No.4, 851-861, October 1966 [320-2910]

FREUDENSTEIN, F. & L. WOO, "Kinematics of the Human Knee Joint," *Bull. of Math. Biophysics*, V.31, No.2, 215-232, June 1969 [320-2928]

FRIEDMAN, H.P. & J. RUBIN, "On Some Invariant Criteria for Grouping Data," *Amer.Statist. Assoc. J.*, V.62, 1159-1178, December 1969 [320-2922]

GORDON, G. & K.ZELIN, "A Study of Emergency Ambulance Service in New York City," *Trans. of the N.Y. Acad.of Sciences, Series II*, V.32, No. 4, 414-427, April 1970 [320-2935]

GORENSTEIN, S., "Printing Press Scheduling for Multi-Edition Periodicals," *Mgmt. Sci.*, V. 16, No. 6, B-373-B-383, February 1970 [320-2958]

GREENSTADT, J., "A Ricocheting Gradient Method for Nonlinear Optimization," *SIAM J. of Appl. Math.*, V.14, No. 3, 429-445, May 1966 [320-2912]

GREENSTADT, J., "On the Relative Efficiencies of Gradient Methods," *Math. of Computation*, V.21, No.99, 360-367, July 1967 [320-2942]

GREENSTADT, J., "Variations on Variable-Metric Methods," *Math. of Computation*, V.24, No.109, 1-19, January 1970 [320-2901]

GRIGORIADIS, M.D., "A Dual Generalized Upper Bounding Technique," *Mgmt. Sci.*, V. 17, No.5, 269-284, January 1971 [320-2973]

GUIGNARD, M., "Generalized Kuhn-Tucker Conditions for Mathematical Programming Problems in a Banach Space," *SIAM J. of Control*, V.7, No.2, 232-241, May 1969 [320-2920]

GUREL, O. & L. LAPIDUS, "Stability Via Liapunov's Second Method," *Ind. & Eng. Chem.*, V.60, No.6, 13-26, June 1968 [320-2906]

GUREL, O. & L. LAPIDUS, "The Maximum Principle and Discrete Systems," *Ind. & Eng. Chem. Fund*, V.7, 617-621, November 1968 [320-2919]

GUREL, O., "Stability of the Pair (X,f)," *Notices of the Amer. Math. Soc.*, V.15, No.5, 774-775, 1968 [320-2957]

GUREL, O. & L. LAPIDUS, "A Guide to Methods for Generation of Liapunov Functions, *Ind. & Eng.Chem.*, V.61, No.3, 30-41, March 1969 [320-2937]

GUREL, O., "Marker Layout Problem Via Graph Theory," *Computing Methods in Optimization Problems-2*, (edited by L.A.Zadeh, L.W. Neustadt & A. V.Balakrishnan), Academic Press, 133-141, 1969 [320-2921]

GUREL, O., "Qualitative Study of Unstable Behavior of Cancerous Cells," *Cancer*, V.24, No.5, 945-947, November 1969 [320-2976]

GUREL, O. & L. LAPIDUS, "Topology of Velocity-Space for Stability Analysis," *Int'l. J. of Control*, V.11, No.1, 19-26, 1970 [320-2961]

HAHN, S., "On the Optimal Cutting of Defective Glass Sheets," *Operations Research*, V.16, No.6, 1100-1114, November-December 1968 [320-2916]

LEMKE, C. & K. SPIELBERG, "Direct Search Algorithms for Zero-One and Mixed Integer Programming," *Operations Research*, V.15, No.5, 892-914, October 1967 [320-2911]

LOOMIS, R. & J. LORENZO, "Experiments in Mapping with a Geo Space Plotter," *Urban & Regional Information Systems for Social Programs*, (edited by John E. Rickert), Center for Urban Regionalism, Kent State University, Ohio, 219-232, 1967 [320-2917]

LOURIE, J., "Loom Constrained Designs: An Algebraic Solution," *Proceedings of the 24th Nat'l.Conf. ACM*, ACM Publication No.P-69, 185-192, 1967 [320-2969]

LOURIE, J., "The Computation of Connected Regions in Interactive Graphics," *Proceedings of the 24th Nat'l. Conf. ACM*, ACM Publication No.P-69, 369-377, 1969 [320-2975]

RAIMOND, J., "Minimaximal Paths in Disjunctive Graphs by Direct Search," *IBM J. of Res. & Development*, V.13, No.4, 391-399, July 1969 [320-2930]

RUBIN, J., "Optimal Classification into Groups: An Approach to Solving the Taxonomy Problem," *J. of Theoret.Biol.*, V.15, 103-144, 1967 [320-2915]

SAVAS, E.S., "Computers in Urban Air Pollution Control Systems," *1) Proceedings of the IBM Scientific Computing Symp. on Water and Air Resource Mgmt., Oct. 23-25, 1967, Yorktown Heights, N.Y.*, 141-173; *2) Socio-Economic Planning Science*, V.1, 157-183, 1967; *3) Ekistics*, 45-64, 1968 [320-2908]

SPIELBERG, K., "Algorithms for the Simple Plant Location Problem with Some Side Conditions," *Operations Research*, V.17, No.1, 85-111, January-February 1969 [320-2900]

SPIELBERG, K., "Plant Location with Generalized Search Origin" Mgmt. *Sci.*, V.16, No.3, 165-178, November 1969 [320-2929]

STRAUSS, C.M. & S. POLEY, "A Three-Dimensional Piping Design Program," *IFIPS Conf. Proceedings, Edinburg, Scotland, 1968*, Information Processing 68- North Holland Publishing Co., Amsterdam, 1431-1440, 1969 [320-2954]

WHITE, W. & A. BOMBERAULT, "A Network Algorithm for Empty Freight Car Allocation," *IBM Sys.J.*, V.8, No.2, 147-169, 1969 [320-2952]

WOO, L., "Type Synthesis of Plane Linkages," *J.Eng. for Ind., Trans. of ASME*, V.89(B), 159-162, February 1967 [320-2953]

WOO, L., "An Algorithm for Straight Line Representation of Simple Planar Graphs," *J. of the Franklin Institute*, V.287, No.3, 197-208, March 1969 [320-2918]

WOO, L. & F. FREUDENSTEIN, "On the Curves of Synthesis in Plane, Instantaneous Kinematics," *Proceedings of the 12th Int'l. Cong. of Appl. Mechnics, Stanford University, Aug. 26-31, 1969*, (edited by M. Hetenyi & W.G. Vincenti), Springer-Verlag, 400-414, 1969 [320-2947]

WOO, L. & F. FREUDENSTEIN, "Application of Line Geometry to Theoretical Kinematics and the Kinematic Analysis of Mechanical Systems," *J. of Mechanisms*, Vol.5, 417-460, 1970 [320-2982]

TECHNICAL REPORTS
(continued)

† ■ 320-2963 *O. Gurel*, The Structure of the Genetic Code, 28 p., January 1969

■ 320-2964 *S. Gorenstein*, Generating Random Normal Numbers, 19 p., February 1969

320-2965 *O. Gurel*, Circular Graph of Marker Layout, 63 p., February 1969

320-2966 *W. White*, Dynamic Transshipment Networks: An Algorithm and Its Application to the Distribution of Empty Containers, 40 p., February 1969

320-2967 *F. Freudenstein, V. Vitagliano, L. Woo & C. Hao*, Dynamic Response of Mechanical Systems, 29+p., March 1969

320-2968 *Proceedings of the Third Annual Technical Management Science/Operations Research Conference*, 206+p., April 1969

† 320-2969 *J. Lourie*, Loom-Constrained Design: An Algebraic Solution, 32 p., April 1969

320-2970 *L. Bodin*, The Catalogue Ordering Problem-2, 33 p., April 1969

† ■ 320-2971 *E. Balas*, Machine Sequencing: Disjunctive Graphs and Degree-Constrained Subgraphs, 31 p., April 1969

■ 320-2972 *J. Colmin & K. Spielberg*, Branch and Bound Schemes for Mixed Integer Programming, 27 p., May 1969

† 320-2973 *M. Grigoriadis*, A Dual Generalized Upper Bounding Technique, 38 p., June 1969

† ■ 320-2974 *L. Bodin & T. Roefs*, Determination of the Operating Policy of a General Reservoir System, 24 p., June 1969

† ■ 320-2975 *J. Lourie*, The Computation of Connected Regions in Interactive Graphics, 26 p., June 1969

† 320-2976 *O. Gurel*, Qualitative Study of Unstable Behavior of Cancerous Cells, 9 p., July 1969

320-2977 *S. Gorenstein*, An Algorithm for Project (Job) Sequencing with Resource Constraints, 60 p., July 1969

320-2978 *G. Moreau & P. Tarbe de Saint Hardouin*, Marker Layout Problem: An Experimental Attempt, 67 p., September 1969

† 320-2979 *C. Lemke, H. Salkin & K. Spielberg*, Set Covering by Single Branch Enumeration with Linear Programming Subproblems, 51 p., October 1969

† 320-2980 *Y. Bard & L. Lapidus*, On Nonlinear System Identification, 21 p., October 1969

320-2981 *O. Gurel*, Algebraic Theory of Scheduling, 32 p., November 1969

† 320-2982 *L. Woo & F. Freudenstein*, Application of Line Geometry to Theoretical Kinematics and the Kinematic Analysis of Mechanical Systems, 103 p., November 1969

† 320-2983 *O. Gurel*, Fundamental Weak Topologies in Living Systems, 18 p., March 1970

320-2984 *H. Hellerman & Y. Ron*, A Time Sharing System Simulation and Its Validation, 36 p., April 1970

320-2985 *O. Gurel*, Functional Groups of Amino Acids and Structural Groups in the Genetic Code, 14 p., April 1970

320-2986 *A. Falkoff & K. Iverson*, The Use of Computers in Teaching Mathematics, 10 p., April 1970

320-2987 O. *Gurel*, Cancer: An Unstable Biodynamic Field, 22 p., April 1970

320-2988 *P. Berry, A. Falkoff & K. Iverson*, Using the Computer to Compute: A Direct But Neglected Approach to Teaching Mathematics, 20 p., May 1970

320-2989 *M. Jenkins*, The Solution of Linear Systems of Equations and Linear Least Squares Problems in APL, 14 p., June 1970

320-2990 *W. W. White*, On the Computational Status of Mathematical Programming, 61 p., June 1970

† 320-2991 *J. Greenstadt*, A Variable-Metric Method Using No Derivatives, 24 p., June 1970

† 320-2992 *M. Grigoriadis*, A Projective Method for a Class of Structured Nonlinear Programming Problems, 142 p., June 1970

320-2993 *W. W. White & E. Wrathall*, A System for Railroad Traffic Scheduling, 101 p., August 1970

320-2994 *K. M. Brown & J. E. Dennis, Jr.*, Derivative Free Analogues of the Levenberg-Marquardt and Gauss Algorithms for Nonlinear Least Squares Approximation, 21 p., August 1970

† 320-2995 *S. Gorenstein*, Programming for Economic Lot Sizes with Precedences between Items - An Assembly Model, 76 p., October 1970

320-2996 *M. Grigoriadis & W. W. White*, A Partitioning Algorithm for the Multicommodity Network Flow Problem, 25 p., October 1970

† 320-2997 *L. H. Scott, E. Wrathall, & S. Poley*, A Railroad Freight Operations Control System, 22 p., December 1970

320-2998 *D. Goldfarb*, Modification Methods for Inverting Matrices and Solving Systems of Linear Algebraic Equations, 58 p., January 1971

320-2999 *S. Gorenstein, S. Poley, & W. W. White*, On the Scheduling of Railroad Freight Operations, 21 p., January 1971

320-3000 *M. Guignard & K. Spielberg*, The State Enumeration Method for Mixed Zero-One Programming, 21 p., February 1971

320-3001 *K. Iverson*, Elementary Algebra, 320 p., June 1971

320-3002 *Z. J. Ghandour*, Formal Systems and Analysis, 28 p., June 1971

320-3003 *J. E. Mezei*, Structure in the Travelling Salesman's Problem The Shklar Algorithm, 30 p., September 1971

320-3004 *J. Rubin*, A Technique for the Solution of Massive Set Covering Problems with Application to Airline Crew Scheduling, 26 p., September 1971

† 320-3005 *T. More, Jr.*, An Interactive Method for Algebraic Proofs, 89 p., September 1971

320-3006 *J. Rubin*, Airline Crew Scheduling - the Non-Mathematical-Problem, 14 p., September 1971

320-3007 *A. D. Falkoff*, A Survey of APL File and IO Systems in IBM, 13 p., November 1971

† 320-3008 *R. H. Lathwell & J. E. Mezei*, A Formal Description of APL, 35 p., November 1971

320-3010 *K. E. Iverson*, APL in Exposition, 66 p., January 1972

320-3011 *M. D. Grigoriadis & W. White*, Computational Experience with a Multicommodity Network Flow Algorithm, 33p., February 1972

320-3012 *W. White*, Mathematical Programming, Multicommodity Flows and Communication Nets, 22p., April 1972

320-3013 *K. Spielberg*, Minimal Preferred Variable Reduction for Zero-One Programming, 21 p., July 1972

320-3014 *K. E. Iverson*, Introducing APL to Teachers, 25 p., July 1972

320-3015 *E. E. McDonnell*, Integer Functions of Complex Numbers, with Applications, 18 p., February 1973

320-3016 *T. More, Jr.*, Notes on the Development of a Theory of Arrays, 83 p., May 1973

320-3017 *T. More, Jr.*, Notes on the Axioms for a Theory of Arrays, 58 p., May 1973

320-3018 *R. Spence*, Resistive Circuit Theory, 191 p., March 1973

320-3019 *K. E. Iverson*, An Introduction to APL for Scientists and Engineers, 26 p., March 1973

320-3020 *P. C. Berry, G. Bartoli, C. Dell'Aquila, & V. Spadavecchia*, APL and Insight: The Use of Programs to represent Concepts in Teaching, 89 p., March 1973

320-3022 *A. D. Falkoff & K. E. Iverson*, Communication in APL Systems, 17 p., May 1973

320-3023 *M. M. Halpern*, Studies in APL: Algebra, Scan, Arithmetic, Permutations, 35 p., June 1973

320-3024 *K. Spielberg*, A. Minimal Inequality Branch-Bound Method, 13 p., June 1973

320-3025 *M. M. Guignard & K. Spielberg*, A Realization of the State Enumeration Procedure, 13 p., June 1973

320-3027 *R. S. Goncharsky, A. Rauch & W. W. White*, Large Scale Mathematical Programming in an APL Environment, 21 p., October 1973

† Appeared in a professional publication, see abstract.

† ■ No copies of either the report or reprint are available from the PSC.

■ Report is out of print. No copies are available from the PSC.

♦ Reprinting of reports originally issued with different numbers.

IBM

IBM Cambridge Scientific Center
545 Technology Square
Cambridge, Massachusetts 02139

IBM Houston Scientific Center
6900 Fannin Street
Houston, Texas 77025

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

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

IBM Philadelphia Scientific Center
3401 Market Street
Philadelphia, Pennsylvania 19104