APL.68000

Reference Card

THE COMPUTER COMPANY
MICRO APL SYSTEMS
1905 Westmoreland Street
Richmond VA 23230-3297
804/358-2171

PRIMITIVE SCALAR MONADIC FUNCTIONS

Function	Description
+Y	Y
-Y	negative of Y (0-Y)
×Y	signum (sign) of Y , represented by ¯1 0 or 1 0
÷Y	reciprocal of Y (1÷Y)
*Y	e to the Yth power
⌈Y	ceiling of Y ; if Y=3.14, 4=⌈Y ; if Y=¯3.14, ¯3=⌈Y
⌊Y	floor of Y ; if Y=3.14, 3=⌊Y ; if Y=¯3.14, ¯4=⌊Y
|Y	absolute value of Y
⍟Y	natural logarithm of Y
!Y	factorial Y ; Gamma function of Y+1
○Y	π times Y
?Y	a random integer from the vector

Note: All scalar functions are extended to operate element-by-element on vectors, matrices, and higher dimensional arrays. Results are displayed if they are not assigned to a variable.

PRIMITIVE SCALAR DYADIC FUNCTIONS

Function	Description
X+Y	adds X to Y
X-Y	subtracts Y from X
X×Y	multiplies X by Y
X÷Y	divides X by Y
X*Y	raises X to power Y
X⌈Y	maximum (larger) of X and Y
X⌊Y	minimum (smaller) of X and Y
X|Y	X residue of Y ; the remainder of Y÷X
X⍟Y	base X logarithm of Y
X!Y	binomial coefficient; for integers X and Y , the number of combinations of Y things taken X at a time

Note: A scalar or one-element array may be used as one argument of a scalar dyadic function and will be reshaped to conform to the size of the other argument.

RELATIONAL FUNCTIONS

Function	Description
X<Y	X less than Y
X≤Y	X less than or equal to Y
X=Y	X equal to Y
X≠Y	X not equal to Y
X≥Y	X greater than or equal to Y
X>Y	X greater than Y

For each relational function, the result is 1 if the relation is true and 0 if the relation is false. Only the = and ≠ functions can be used with character data. A scalar or one element array may be used as one argument to relational functions and will be reshaped to conform to the size of the other argument.

LOGICAL FUNCTIONS

Function	Description	if	X	= 0 0 1 1
		and	Y	= 0 1 0 1
X∧Y	X and Y are true	then	X∧Y	= 0 0 0 1
X∨Y	X or Y is true, or both		X∨Y	= 0 1 1 1
X⍲Y	not both X and Y are true		X⍲Y	= 1 1 1 0
X⍱Y	neither X nor Y is true		X⍱Y	= 1 0 0 0
~Y	not Y (logical negation)		~Y	= 1 0 1 0

Logical functions can be used only when both X and Y consist solely of 0 s and 1 s. Possible results of logical functions are given in the table above. A scalar or one-element array may be used as one argument to logical functions and will be reshaped to conform to the size of the other argument.

MIXED MONADIC FUNCTIONS

Function	Description
⍴Y	Shape (dimension vector) of Y
⍳Y	Vector of first Y integers, beginning with the origin (0 or 1)
,Y	Ravel of Y ; make Y a vector
⊖Y	Reversal along the first axis of Y
⌽Y	Reversal along the last axis of Y
⌽[Z]Y	Reversal along the Zth axis of Y
⍉Y	Monadic transpose; reverses order of all coordinates of Y
⍋Y	Sort up; the permutation which subscripts Y in ascending order
⍒Y	Sort down; the permutation which subscripts Y in descending order
X[Y]	The elements of X selected by index Y
⌹A	Left inverse of matrix A ; least squares solution of I=A+.×X , where I is an identity matrix
⍕Y	Monadic format; returns a character array that looks like Y when printed

Note: Mixed functions are those in which the shape of the result may be different from the shapes of the arguments.

MIXED DYADIC FUNCTIONS

Function	Description
X⍴Y	Reshape Y to the specification of dimension vector X
X⍳Y	The index of the first occurrence of Y in vector X
X∊Y	Membership; is X included in Y . Result is a logical array with the same shape as X ; 1 signifying membership, 0 signifying not
X?Y	Random deal. X integers chosen without replacement from ⍳Y
X⊤Y	Encode. Representation of Y in number system with radix X
X⊥Y	Decode. Value of the representation Y in number system with radix X
X,Y	Catenate. X and Y catenated along last axis; if X and Y are scalars, a vector is formed
X,[Z]Y	Catenate. X and Y catenated along Z th axis; either X or Y may have one axis more than the other; Z applies to the larger
X,[Z.5]Y	Insert a new dimension between axes Z and Z+1
X⊖Y	Rotation by X elements along the first axis of Y
X⌽Y	Rotation by X elements along the last axis of Y
X⌽[Z]Y	Rotation by X elements along the Z th axis of Y
X⍉Y	Transpose by X of the coordinates of Y
X↑Y	Take. If X positive, take first X elements of Y and pad with zeros or ones if necessary. If X negative, take last X elements of Y and pad with zeros or ones if necessary.
X↓Y	Drop. If X positive, drop first X elements of Y . If X negative, take last X elements of Y .
B⌹A	Least squares solution of B=A+.×X for matrix A , vector B , or columns of matrix B .

Note: Mixed functions are those in which the shape of the result may be different from the shapes of the arguments.

FORMATTING FUNCTIONS

Function	Description
X⍕Y	The Format function converts the elements of numeric array Y to a character representation of Y based on the specifications of X , which usually has two columns. Some examples are as follows:

```
12 2⍕Y  Column width = 12, 2 decimal places
14 0⍕Y  Column width = 14, 0 decimal places
  0 2⍕Y  Column width determined by data,
         2 decimal places
     4⍕Y  Column width determined by data,
         4 decimal places
 8 ¯3⍕Y  Column width = 8, 3 significant digits,
         scientific notation
```

If X has more than two elements, each pair of elements in X specifies the format for each column of Y .

Function	Description
X⍺Y	The Alpha function converts the elements of numeric array Y to a character representation of Y based on the specifications of X . Alpha can be used in a way similar to Format, as shown in the first example below; it can also be used with a COBOL-like picture clause, as shown in the second example.

```
      7 2 7 3⍺¯123.456 78.9012
¯123.46  78.901
      '$$Z,ZZ9.99 CR'⍺¯1234.56 123.78
$1,234.56 CR   $123.78
```

OPERATORS

Primitive scalar dyadic functions may be incorporated into specialized composite functions as shown below. In the notation used below, both f and g may be any primitive scalar dyadic function (+ - × ÷ ⋆ ⌈ ⌊ | ⍟ !).

Function	Description
	Reduction. Reduce Y right to left, inserting f between each element. Returns result having one less dimension than Y. For example: -/1 2 3 4 is equivalent to (1-(2-(3-(4)))) which yields a result of ¯2
f/Y	Reduction along last dimension of Y
f⌿Y	Reduction along first dimension of Y
f/[Z]Y	Reduction along Z th dimension of Y
	Scan. Scan Y left to right, performing reductions with f on progressively longer sub-arrays of Y. The result has the same shape as Y. For example: +\3 2 1 3 5 6
f\Y	Scan along last dimension of Y
f⍀Y	Scan along first dimension of Y
f\[Z]Y	Scan along Z th dimension of Y
Xf.gY	Generalized inner product of X and Y
X∘.gY	Generalized outer product of X and Y

COMPRESSION AND EXPANSION

Function	Description
	Logical compression by bit vector X along any array Y.
X/Y	Compression along the last axis of Y
X⌿Y	Compression along the first axis of Y
X/[Z]Y	Compression along the Z th axis of Y
	Logical expansion by bit vector X along any array Y.
X\Y	Expansion along the last axis of Y
X⍀Y	Expansion along the first axis of Y
X\[Z]Y	Expansion along the Z th axis of Y

TRIGONOMETRIC FUNCTIONS

Circular and hyperbolic functions and their inverses, where Y is in radians and X is within the range of the function:

Function	Description	Function	Description
0○Y	(1-Y⋆2)⋆0.5		
1○Y	sin Y	¯1○X	arcsin X
2○Y	cos Y	¯2○X	arccos X
3○Y	tan Y	¯3○X	arctan X
4○Y	(1+Y⋆2)⋆0.5	¯4○X	(¯1+X⋆2)⋆0.5
5○Y	sinh Y	¯5○X	arcsinh X
6○Y	cosh Y	¯6○X	arccosh X
7○Y	tanh Y	¯7○X	arctanh X

FUNCTION EDITING

∇	Open or close function definition mode
⍫	Lock function code
[⎕]	Display entire function
[⎕N]	Display function beginning with line N
[N.M]	Insert line N.M
[N]	Replace line N
[N⎕]	Display line N
[N⎕M]	Display line N and place cursor under position M for line editing. A / removes a character, and a digit 1-9 or a letter causes spaces to be inserted into which changes can be inserted; in this scheme an A=5 spaces, B=10 spaces, etc.
[∆N]	Deletes line N of the function

SPECIAL SYMBOLS

Symbol	Description
X←Y	Assignment Statement. The value of Y is assigned to a variable named X ; Y may be any APL expression
X←⎕	Request Input. The value of the expression entered is assigned to X ; → or Ø provides escape from request.
X←⍞	Request Character Input. The value of input text up to but not including the carriage return is treated as literal characters and is assigned to X ; Ø provides escape from request.
⎕←X	Print the value of expression X and return carriage.
⍞←X	Print the value of expression X and do not return carriage.
'XYZ'	The literal characters XYZ ; anything entered between the quotes will be evaluated as a literal expression, with '' being interpreted as a single quote.
Ø	The OUT character, made by overstriking the letters O, U, and T, provides an escape from the request for input ⎕: and ⍞: .
[ ]	Brackets are used to enclose indexing coordinates for the array or operator defined to its immediate left.
( )	Parentheses are used to define explicitly the order of execution; function execution is right to left, except as modified by parentheses; expressions may be of any complexity desired.
→X	Branch to line number X, where X is a scalar, or to the first element of X where X is a vector. If X is an empty vector, continue processing with the next statement. If X is not in the range of statement numbers in the function, exit the function.
→	Terminate execution of this function or of the last suspended function, and all those functions which led to its use. → is also used to escape from ⎕:
_	Underline is used in constructing the characters A̲ through Z̲ and ⍙, which are the only valid underlined characters in APL.
⍝	Comment. This character in a statement indicates that the remainder of the line is a comment and is not to be executed. Any valid characters may be used in the comment text.
∇	The DEL character declares a change from execution mode to function definition mode; a second DEL terminates function definition.
◊	A diamond is used to separate statements entered on the same line; the leftmost statement is processed first. ⋄ and ⋄ may also be used.
:	Colon. Separates a statement label from the statement in a line of a function, e.g. L1: X←1
;	Semi-Colon. Separates index coordinates when subscripting matrices and higher order arrays.
⍎	Execute function executes a literal expression as if it were unquoted, for example, ⍎'2+3'↔5

OVERSTRUCK CHARACTERS

Character		Made With	Character		Made With
{ }	Braces	[∘] ∘	⍟	Logarithm	○ ⋆
⍀	Column Backslash	\ -	⌹	Matrix Division	⎕ ÷
⌿	Column Slash	/ -	⍲	Nand	∧ ~
!	Combination	' .	⍱	Nor	∨ ~
⍝	Comment	∩ ∘	Ø	OUT	O U T
⋄ ⋄	Diamond	< > ∨ ∧	⍫	Protected Function	∇ ~
$	Dollar Sign	S |	⍞	Quote-Quad	⎕ '
⍎	Execute	⊥ ∘	⌽	Reversal	○ |
!	Factorial	' .	⌽	Rotate	○ |
⍗	File Drop	⎕ ↓	⍒	Sort Down	∇ |
⍐	File Hold	⎕ ↑	⍋	Sort Up	∆ |
⍇	File Read	⎕ ←	⊣ ⊢	Tack	(-) -
⍈	File Write	⎕ →	⊖	Theta	○ -
⍕	Format	⊤ ∘	⍉	Transpose	○ \
⌶	I-Beam	⊥ ⊤	A̲	Underlined letter	A _

SYSTEM FEATURES

Key to Notation
A = Array of Data
B = Scalar, 0 or 1
C = Character Vector
F = Function Name
M = Character Matrix
N = List of Names
S = One Element Numeric
V = Vector

Feature — Description

⎕A .... Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ

⎕A̲ .... Alphabet underlined: A̲B̲C̲D̲E̲F̲G̲H̲I̲J̲K̲L̲M̲N̲O̲P̲Q̲R̲S̲T̲U̲V̲W̲X̲Y̲Z̲

⎕AI .... Account Information: account number, CPU time in milliseconds, connect time in milliseconds, keyboard unlock time in milliseconds, database access units, month-to-date CPU time, month-to-date connect time

⎕B .... Backspace, lower case

⎕B̲ .... Backspace, upper case

C ⎕BOX C .... ⎕BOX restructures a vector into a matrix, breaking according to the delimiter element, and pads each row with the fill element, for example:

```
        ⎕←A←'|o' ⎕BOX 'APRIL|MAY|JUNE'
APRIL
MAYoo
JUNEo
```

⎕BOX restructures a matrix into a vector, eliminating trailing fill characters from each row and separating each by the delimiter character, for example:

```
        '*o' ⎕BOX A
APRIL*MAY*JUNE
```

⎕BOX C .... If the left argument is omitted, ⎕BOX defaults to blanks for character data and to zeros for numeric data.

⎕C .... Control Characters (ASCII): ⎕C[⎕IO+7], for example, is a bell character.

⎕CR F .... Canonical representation of a function; F is a character vector containing the function name.

⎕CT .... Comparison Tolerance (relative). ⎕CT is used in ⌈⌊<≤=≠≥>∊⍳ operations and may be changed by assignment.

⎕D .... Digits: 0123456789

⎕DBR C .... Delimited Blank Removal. Removes all leading and trailing blanks and compresses embedded multiple blanks to one blank, for example:

```
        ⎕DBR ' THIS    IS       IT.  '
THIS IS IT.
```

C ⎕DBR C .... If a left argument is used, all blanks preceding and following delimiters specified in the left argument are eliminated, for example:

```
        ',:' ⎕DBR ' THIS .  IS : IT  ! '
THIS,IS:IT  !
```

⎕DL S .... Delay Execution. Delays execution for the number of seconds specified; returns the number of seconds actually delayed.

⎕DR A .... Data Representation. Returns a number which indicates the data type of A, according to the following code:
1 = Boolean
2 = Integer
3 = Floating Point
4 = Character

S ⎕DR A .... When used with a left argument which is 1, 2, 3, or 4, ⎕DR converts A to the data type specified, for example,

```
        4 ⎕DR DATA
```

converts DATA to character data.

⎕ERX S .... Error Trapping. Sets error trap and transfers control to line number S when an error is encountered; the function returns the previous trap value.

⎕EX N .... Expunge. Erase name(s) contained in the character vector or character matrix of APL object names. Returns a 1 if object erased, a 0 if not.

SYSTEM FEATURES (Continued)

Feature — Description

⎕FX M ....
B ⎕FX M .... Fix. Establish a function in the active workspace from the character matrix specified; function returns the name of the fixed function. A left argument of 1 causes the fixed function to be locked; a left argument of 0 causes the fixed function to be unlocked.

⎕HC S .... Hard Copy. Write hard copy to output device number specified.

⎕I .... Idle, lowercase (time-fill character)

⎕I̲ .... Idle, uppercase.

⎕IO .... Index Origin. Affects indexing, ? and ⍳.

⎕L .... Line Feed.

⎕LC .... Line Counter. Line number of functions in execution, innermost first.

⎕LER .... Line/Error Report. Returns an error code and the line number for the last error; see error code table on this reference card for a list of codes.

⎕LX .... Latent Expression. Expression executed on loading a workspace.

⎕M .... Months. Names of the months of the year.

⎕MOUNT V or M ... Mount. Assign devices to logical unit numbers 0-9, for example:

```
        ⎕MOUNT 5 7
```

assigns device 5 to logical unit 0 and
assigns device 7 to logical unit 1.

⎕N .... Non-printing Null.

⎕NC N .... Name Classification. Right argument can be a line label, variable, function, or group. Name classification is 1, 2, 3,or 5, respectively.

⎕NL V ....
C ⎕NL V .... Name List. List of name(s) in sorted order of object(s) in the name classes specified in the right argument. A left argument may be used to restrict the list to object(s) beginning with the character(s) specified, for example:

```
        CV ⎕NL 3
```

produces a sorted list of functions (name class 3) whose names begin with letters in character vector CV.

⎕PP .... Print Precision; number of digits in numeric output and monadic ⍕ (format); may be changed by assignment.

⎕PW .... Print Width. Affects all output except ⍞.

⎕R .... Return Carriage.

⎕RL .... Random Link. Used in random number generator (?); may be changed by assignment.

⎕SS (C;C) .... String Search and Replace. For example:

```
        A←'MARY HAD A LITTLE LAMB'
        ⎕SS (A;'LITTLE')
12
```

The string LITTLE begins at position 12.

⎕SS (C;C;C) ....

```
        ⎕SS (A;'LITTLE';'BIG')
MARY HAD A BIG LAMB
```

The string LITTLE is replaced by BIG .

V ⎕STOP F ......
⎕STOP F .... Stop Vector. The left argument is a vector of statement numbers in function F at which execution is suspended. Without a left argument, the stop vector is returned.

⎕T .... Tab, horizontal.

⎕TS .... Time Stamp. Year, month, day, hour, minute, second, and millisecond.

⎕TT .... Terminal Type. Returns terminal type as a code number.

V ⎕TRACE F .....
⎕TRACE F .... Trace. Print value of assignments made on the statement numbers specified in the left argument. Omitting the left argument returns the trace vector of statement numbers.

⎕UL .... User Load. Number of users signed on the system.

⎕W .... Week. Names of the days of the week.

⎕WA .... Work Area. Number of bytes remaining in the workspace.

ERROR MESSAGES

Message	Problem and Corrective Action
BUFFER FULL	Input line too long; use backspace and linefeed to shorten the line.
CHARACTER	Improper overstrike; Reenter overstrike.
COMPONENT NOT IN FILE	The file does not contain the specified component.
COPY BUFFER FULL	Name list of)COPY command is too long; shorten name list or)GROUP all or some of the names.
DEFN	Improper attempt at function definition, function line editing, or syntax of function header improper as a result of header editing, function is pendant, or function is locked. An object already has the name; erase it or choose another name for the function. During interrupted function execution, clear the state indicator.
DISK FULL	File data space is full.
DOMAIN	Function not defined for the arguments given; provide proper arguments.
FILE ALLOCATION EXCEEDED	The file has reached its maximum allowed size.
FILE IN EXISTENCE	Attempt to rename a file which already exists.
FILE LOCKED	An incorrect file password has been used.
FILE MAINTENANCE IN PROGRESS	The file system is temporarily not available.
FILE NOT IN SYSTEM	The file does not exist in the file system.
FILE OR COMPONENT HELD	The operation cannot be performed due to an outstanding file or component hold by another user.
INCORRECT COMMAND	Spelling or syntax of command is faulty; reenter correctly.
INDEX	Reference to nonexistent element of an array; provide proper indices.
I/O ERROR	APL encountered an error during tape or disk input/output; probable hardware failure.
LENGTH	Arguments are of unequal lengths; ensure arguments are of equal lengths or that proper coordinates are referenced.
NONCE	Syntax for operation not supported; reformulate the expression.
NOT COPIED: object(s)	Attempt to)PCOPY an object which exists in the active ws; check spelling of workspace or object name.
NOT FOUND: object(s)	Workspace does not contain the object (variable, function, or group).
NOT GROUPED, NAME IN USE	Variable or function already has the name; change name of group or erase conflicting object.
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 using the)WSID command.

ERROR MESSAGES (Continued)

Message	Problems and Corrective Action
NOT SAVED, WS LOCKED	An established workspace cannot be saved unless the active workspace has the same lock; change the lock of the active workspace using the)WSID command.
NOT WITH OPEN DEFINITION	Command cannot be processed while user is in definition mode; close the definition with a ∇ .
RANK	Function not defined for array(s) of this structure; provide argument of correct structure (scalar, vector, or matrix).
SI DAMAGE	A pendant or suspended function was replaced or removed by)COPY or)ERASE commands. Label lines of a suspended function are edited or a function not at the top of the SI list is edited, erased, or copied or a function on the SI list has its header edited. Action: Clear the state indicator by)SICLEAR .
SYMBOL TABLE FULL	Too many names used for the current symbol table size. Action: Save current ws, clear, increase size of symbol table with)SYMS and copy saved ws into active ws.
SYNTAX	Ill-formed expression or incorrect number of arguments for a function. Action: Reformulate statement.
SYSTEM	A problem internal to APL. No action necessary; a clear ws is loaded automatically. If error recurs, contact The Computer Company.
UNAUTHORIZED FILE ACCESS	The file's access matrix does not allow the operation from this user number. Action: Modify access matrix.
USER ALLOCATION EXCEEDED	User has too many files or the aggregate size of all files exceeds the user's quota.
VALUE	Value for this name not previously specified or the explicit result of a defined function was not specified during execution. Action: Specify a value for the indicated variable.
WS FULL	Insufficient working area. The active ws cannot contain all the objects requested. During a)COPY command, no objects are copied. Action: Erase any objects no longer required. Reduce the size of the symbol table. Clear the state indicator. Reduce the size of the symbol table. Clear the state indicator.
WS LOCKED	Lock incorrect, omitted, or workspace has no lock.
WS NOT FOUND	The workspace with that name is not in the specified library or logical unit. Action: Check location of workspace and the spelling of the workspace name.

FILE OPERATIONS

Key to Notation

R = Result
F = File Number
C = Component Number (0 is assumed if omitted)
U = User Number (⎕AI[1] is assumed if omitted)
Q = Allocation Quota
P = File Password (0 is assumed if omitted)
[ ] = Enclosed is optional

⊟ Write

R←A⊟F,C [,U,P] Writes the expression or variable A to the indicated file and component.
If C=0 then A is appended to the end of the file.
If C=N then A replaces component N .
If C=N.5 then A is inserted before component N+1.

⊟ Read

R←⊟F,C [,U,P] Reads the contents of component C from the indicated file.

⊞ Delete

R←⊞F,C [,U,P] Deletes component C from the indicated file.
R←U⊞F,0 [,U,P] Deletes file F from system.
If the delete operation was successful, a 1 is returned; otherwise, a 0 is returned.

⊞ Hold/Release

R←X⊞F,C [,U,P] Hold/Release a component of the indicated file, according to the value of X .
If X=0 then release the component/file.
If X=1 then hold— restrict write access by other users.
If X=2 then hold— restrict read and write access by other users.
If C=0 then the whole file is held or released.
If the operation was successful, a 1 is returned; otherwise, a 0 is returned.

Set File Allocation Quota

R←⊞F,Q [,U,P] Set file allocation quota.
R←⊞0,Q Set default file allocation.
R←2⊞0 0 Returns user quota vector of:
1 - User number
2 - Aggregate file allocation quota
3 - Current aggregate file size
4 - Number of files quota
5 - Number of files in existence
6 - Allocation assigned to a new file

Rename/Replace

R←(Fnew,0,Unew,Pnew)⊟F,0,U,P Rename file, change password.
R←(Fnew,64,Unew,Pnew)⊟F,0,U,P Replace file Fnew by F .

Information Read

R←1⊟F,0 [,U,P] Returns the number of components in the file.
R←2⊟F,0 [,U,P] Returns file description, a nine element vector consisting of:
1 - File number
2 - Maximum allowed size in bytes
3 - Actual size
4 - Number of components
5 - Date file was created, MMDDYY
6 - Time file was created, HHMMSS
7 - Date file was last updated, MMDDYY
8 - Time file was last updated, HHMMSS
9 - Bytes attributable to file overhead
R←3⊟F,C [,U,P] Returns component description, a six element vector of:
1 - File number
2 - Component number
3 - User number
4 - Date component was written, MMDDYY
5 - Time component was written, HHMMSS
6 - Size of component, in bytes
R←5⊟0 Returns list of files owned by user.

FILE OPERATIONS (Continued)

R←6⊟F,0 [,U,P] Returns file hold description, a three element vector of:
1 - Number of components held
2 - User holding the file (or 0)
3 - Hold restriction (0, 1, or 2)
R←7⊟F,C [,U,P] Returns component hold description, a three element vector of:
1 - Component number
2 - User holding the component (or 0)
3 - Hold restriction (0, 1, or 2)

The Access Matrix

R←⊟-F Reads the access matrix.
R←A⊟-F Writes the access matrix.

The access matrix is an N by 2 matrix in which each row is a user number followed by a number which is the sum of the access codes that the user is authorized. The default access matrix grants the owner all privileges except file delete. The access codes are listed below.

Access Code	Operation	Access Code	Operation
1	Read file components	512	Set file allocations
2	1⊟, 2⊟, 3⊟	1024	Rename file
4	Insert components	2048	Hold/Release file
8	Append components	4096	Hold/Release components
16	Replace components	16384	6⊟, 7⊟
64	Delete a file	524288	Read access matrix
128	Delete components from a file	1048576	Write access matrix

OVERLAYS

)COPY SYSFNS ⍙OV

R←0 ⍙OV Name-Matrix Create overlay. R will be an overlay of all objects whose names are in Name-Matrix.

R←1 ⍙OV Overlay Disperse overlay with all functions locked. R will be a matrix of names of all dispersed objects.

R←2 ⍙OV Overlay Disperse overlay, leaving all unlocked functions unlocked. R will be a matrix of names of all dispersed objects.

ERROR TRAP RETURN CODES

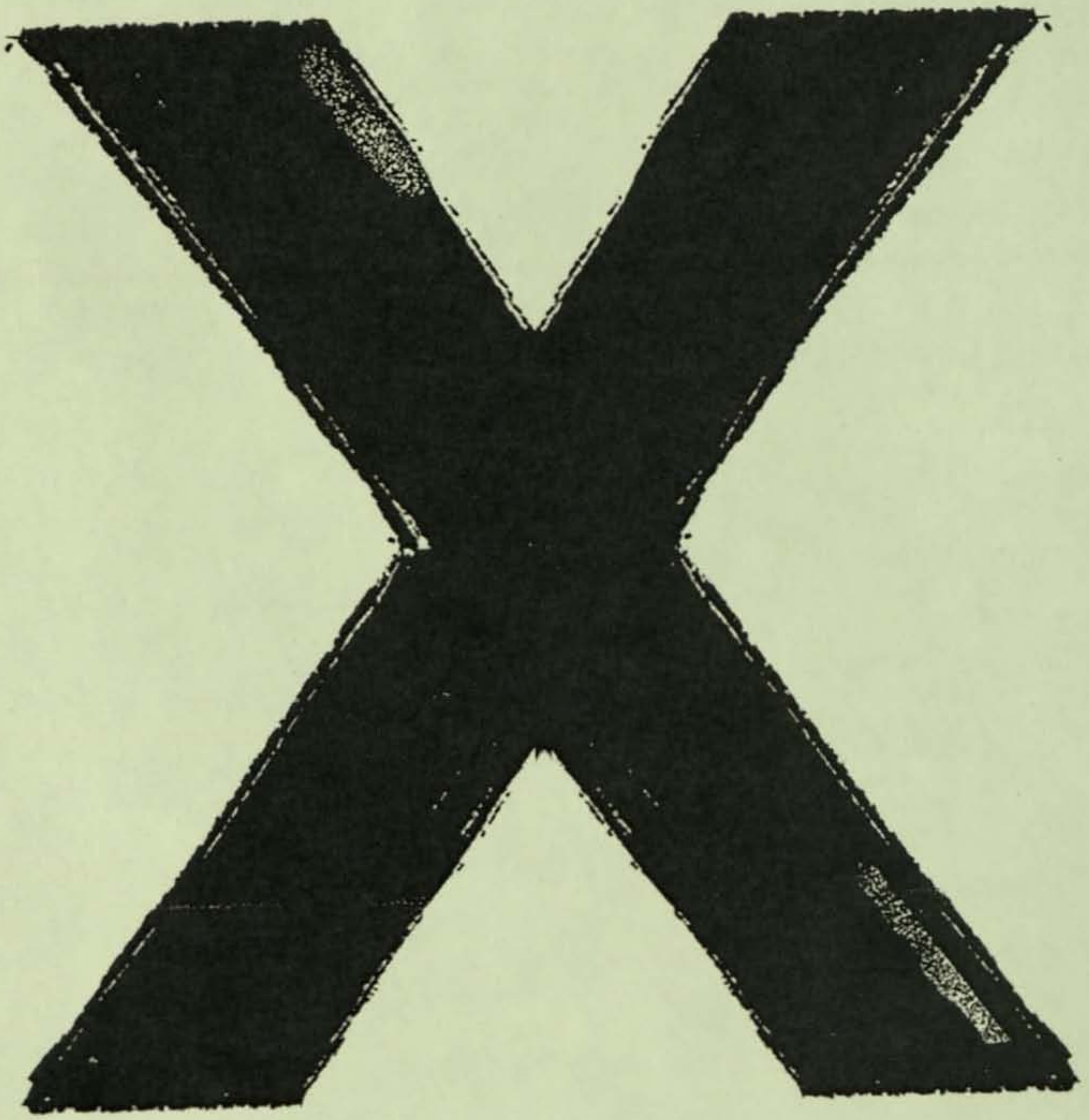
Error Number	Description
1	WS FULL
2	SYNTAX ERROR
3	INDEX ERROR
4	RANK ERROR
5	LENGTH ERROR
6	VALUE ERROR
11	DOMAIN ERROR
13	INTERRUPT
16	NONCE ERROR
17	FILE I/O ERROR
18	FILE NOT IN SYSTEM
19	UNAUTHORIZED FILE ACCESS
20	COMPONENT NOT IN FILE
21	FILE ALLOCATION EXCEEDED
23	FILE MAINTENANCE IN PROGRESS
24	FILE OR COMPONENT HELD
25	INCORRECT COMMAND
26	DATA DAMAGED
27	USER NOT IN SYSTEM
28	USER ALLOCATION EXCEEDED
29	FILE IN EXISTENCE
40	DISK FULL
41	FILE LOCKED
42	LOGICAL UNIT NOT FOUND

SYSTEM COMMANDS

Key to Notation

[ ]	= enclosed argument is optional
ws	= workspace
wsid	= workspace identification
name	= name of variable, function, or group
ltr	= letter used to begin listing
lu	= logical unit
(s)	= one or more

Command and Arguments	Description
)CLEAR	Clear the active ws
)CONTINUE	End work session and store the active ws under the name CONTINUE
)COPY [lu]wsid[name(s)]	Copy selected objects from a stored ws into the active ws
)DIGITS 1 to 15	Set maximum number of significant digits for output
)DIGITS	List current setting for DIGITS
)DROP [lu]wsid	Remove the wsid specified from the library
)ERASE name(s)	Remove any global object from the active ws; remove functions, variables, and groups, including objects in groups
)FNS [ltr]	List the names of all defined functions, or those beginning with the letter specified
)GROUP name(s)	Gather functions and variables into a group; first name is name of group
)GRP name	List the names of objects in the specified group
)GRPS [ltr]	List the names of groups, or those beginning with the letter specified
)LIB [lu] [ltr]	List the names of all workspaces in a specified library, or those beginning with the letter specified
)LOAD [lu] wsid	Replace active ws with a copy of a stored ws
)OFF	End a work session
)ORIGIN 0 or 1	Set index origin to 0 or 1
)ORIGIN	List current ORIGIN setting
)PCOPY[lu]wsid[name(s)]	Copy selected objects from a stored ws into the active ws if the names are not in the active ws
)SAVE	Store the active ws under its current wsid
)SAVE [lu]wsid	Store a copy of the active ws under the ws specified
)SCOPY[lu]wsid[name(s)]	Silent COPY ; same as)COPY , except no message is printed upon execution
)SDROP[lu]wsid	Silent DROP ; same as)DROP , except no message is printed upon execution
)SI	State Indicator; list the names of all halted functions
)SIV or)SINL	List the names of all halted functions and associated local variables
)SICLEAR	Clear the State Indicator
)SLOAD[lu]wsid	Silent LOAD ; same as)LOAD , except no message is printed upon execution
)SPCOPY [lu]wsid[name(s)]	Silent PCOPY ; same as)PCOPY , except no message is printed upon execution
)SSAVE	Silent SAVE ; same as)SAVE , except no message is printed upon execution
)SSAVE[lu]wsid	Silent SAVE ; same as)SAVE , except no message is printed upon execution
)SWSID[lu]wsid	Silent WSID ; same as)WSID , except no message is printed upon execution
)SYMB 1 to 6000	Set size of symbol table
)SYMB	List current number of symbols in the symbol table
)TABS 1 to 160	Set tabs for automatic tabbing for input and output; tabs must correspond to terminal tab settings
)TABS	List current tab positions
)VARS [ltr]	List the names of all global variables, or those beginning with the letter specified
)WIDTH 40 to 390	Set maximum number of positions in a line of output
)WIDTH	List the current WIDTH setting
)WSID [lu]wsid	Rename the active ws
)WSID	List wsid of active ws


X

APL GRAPH-II

USERS' REFERENCE CARD

This document is a summary of
PLOT-10/APL GRAPH-II
Standard Function Package—
TEKTRONIX Part No. 062-1617-01
Implementation for APL\360—
TEKTRONIX Part No. 062-1618-03

For further information, contact your TEKTRONIX Applications Engineer.

TEKTRONIX

Information about syntax illustrations:

- Upper case letters, numbers, parentheses, and punctuation marks must be coded as shown. Never code brackets, braces or ellipses.
- Lower case items represent variables as follows:
  a—the explicit result of a function.
  n—a scalar expression.
  v—a vector expression.
- Other lower case letters clearly represent scalar or vector expressions.

```
[* Items or groups of items within brackets are          ]
[  optional; code one or none.                           ]
{* Braces group alternative items; code one.             }
```

GENERAL ROUTINES

ANMODE	Put terminal in Alpha-numeric mode and dump output buffer.
BELL n	Sound audible tone 'n' times.
HDCOPY n	Make 'n' hardcopies of current information on screen.
INITT	Initialize system variables.
ERASE	Erase the screen.
RESTAT v	Restore the system status to 'v'.
a ←*SVSTAT*	Save the current system status in 'a'.

All general routines are present in *MINI APL GRAPH-II.*

ENVIRONMENTAL ROUTINES

Special Environmental Routines

SET v — The leftmost function on a line. Used to discard the explicit result, if not needed, of an environmental parameter setting routine, e.g.

SET ROTATION TO 45

a←*TO* v — Used for an absolute specification as opposed to a relative one, e.g.

```
SET ROTATION TO 45   is absolute, while
SET ROTATION 45      is relative.
```

Relative specifications are additive except for *SCALE*, which is multiplicative.

DEFN — If used as the parameter to an environmental routine, no change will be effected on the environment, but the current value is returned, e.g.

A←ROTATION DEFN

will place in *A* the current rotation parameter.

Parameter Setting Routines

Note: The following routines have an explicit result which is the effect on the current environmental parameter. This value may be discarded by the use of the *SET* function.

Set character size for 4015:

```
a←CHARSIZE {[TO] n}
           {DEFN  }
```

†Set length of software dashed line segments:

```
a←DASHLENGTH {[TO] length}
             {DEFN       }
```

Set rotation factor for relative lines:

```
a←ROTATION {[TO] degrees}
           {DEFN        }
```

Set scale factors for relative lines:

```
a←SCALE {[TO] xscale [,yscale]}
        {DEFN                 }
```

†Set window and viewport definitions:

```
   {VIEWPORT}  {[{CORNER}[TO]  xmin,ymin]
a←{         }  {[{CENTER}                ]
   {WINDOW  }  {  DEFN

                      [,SIZE [TO] xsize,ysize]}
                                              }
```

Environment Specification

Set the graphic environment. Default environment, set by *INITT*, is underlined. *VIRTUAL* is not valid in *MINI APL GRAPH-II.*

```
USING {[SCREEN     ][.APL  ][.BUFFERED]
      {[INCHES     ][ASCII ][DIRECT   ]
      {[CENTIMETERS]
      {[CHARACTERS ]
      {[†VIRTUAL   ]

                            [.4013][.FOCUS  ]}
                            [4015 ][DEFOCUS ]}
                                             }
```

FUNCTIONAL ROUTINES

Graphic Input

a ← *CURSOR* — Returns decimal ASCII/*APL* equivalent of the keyed in character and the X and Y location of crosshairs, as positioned by the user, according to the current environment.

Graphic Output

All of the graphic output functions can take the form:

```
operation [TO]{x1 y1 x2 y2 ••• xn yn       }[AT  xloc,yloc]
              {x1 x2 ••• xn WITH y1y2 ••• yn}
```

Where 'operation' can be:

† *DASH*	Software dashed lines
† *DOT*	EGM dot lines
† *DOTDASH*	EGM dot-dash lines
DRAW	Bright lines
† *LONGDASH*	EGM long dashed lines
MOVE	Dark line
† *POINT*	Point plot mode
† *SHORTDASH*	EGM short dashed lines

The presence or absence of *TO* indicates that the X and Y coordinates to follow are absolute or relative.

The coordinates are specified as either X-Y pairs or by using the *WITH* function to combine logically X and Y as separate variables. *AT* is used for a *MOVE TO* xloc yloc, according to the current environment.

Character Output

STRING 'character-string' [*AT* xloc, yloc]

Output a string of characters at the current beam location.

RESERVED VARIABLES

The following is a list of the variables which are reserved for use by *APL GRAPH-II.* Their values should not be altered by a user's program.

```
APL         ASCII      BUFFERED              CENTIMETERS
CHARACTERS             DEFN       DEFOCUS    DIRECT       FOCUS
INCHES      SCREEN     VIRTUAL    BUF        COD          CSZ
CVT         ROT        SCL        S1         S10          S11
S12         S13        S14        S15        S16          S17
S2          S3         S4         S5         S6           S7
S8          S9         WIN
```

†These routines are not present in *MINI APL GRAPH-II.*

APL CODE CHART

CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
NUL Ø	DLE 16	SP 32	0 48	¯ 64	⋆ 8Ø	⋄ 96	*P* 112
SOH 1	DC1 17	¨ 33	1 49	α 65	? 81	*A* 97	*Q* 113
STX 2	DC2 18	) 34	2 5Ø	⊥ 66	ρ 82	*B* 98	*R* 114
ETX 3	DC3 19	< 35	3 51	∩ 67	⌈ 83	*C* 99	*S* 115
EOT 4	DC4 2Ø	≤ 36	4 52	⌊ 68	~ 84	*D* 1ØØ	*T* 116
ENQ 5	NAK 21	= 37	5 53	∈ 69	↓ 85	*E* 1Ø1	*U* 117
ACK 6	SYN 22	> 38	6 54	_ 7Ø	∪ 86	*F* 1Ø2	*V* 118
BEL 7 BELL	ETB 23	] 39	7 55	∇ 71	ω 87	*G* 1Ø3	*W* 119
BS 8 BACK SPACE	CAN 24	∨ 4Ø	8 56	∆ 72	⊃ 88	*H* 1Ø4	*X* 12Ø
HT 9	EM 25	∧ 41	9 57	ι 73	↑ 89	*I* 1Ø5	*Y* 121
LF 1Ø LINE FEED	SUB 26	≠ 42	(58	∘ 74	⊂ 9Ø	*J* 1Ø6	*Z* 122
VT 11	ESC 27	÷ 43	[59	' 75	← 91	*K* 1Ø7	{ 123
FF 12	FS 28	, 44	; 6Ø	⎕ 76	⊢ 92	*L* 1Ø8	⊣ 124
CR 13 RETURN	GS 29	+ 45	× 61	| 77	→ 93	*M* 1Ø9	} 125
SO 14	RS 3Ø	. 46	: 62	⊤ 78	≥ 94	*N* 11Ø	$ 126
SI 15	US 31	/ 47	\ 63	○ 79	- 95	*O* 111	127 RUBOUT (DEL)

ASCII CODE CHART

CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
NUL Ø	DLE 16	SP 32	Ø 48	@ 64	P 8Ø	` 96	p 112
SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
STX 2	DC2 18	" 34	2 5Ø	B 66	R 82	b 98	r 114
ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
EOT 4	DC4 2Ø	$ 36	4 52	D 68	T 84	d 1ØØ	t 116
ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 1Ø1	u 117
ACK 6	SYN 22	& 38	6 54	F 7Ø	V 86	f 1Ø2	v 118
BEL 7 BELL	ETB 23	' 39	7 55	G 71	W 87	g 1Ø3	w 119
BS 8 BACK SPACE	CAN 24	(4Ø	8 56	H 72	X 88	h 1Ø4	x 12Ø
HT 9	EM 25	) 41	9 57	I 73	Y 89	i 1Ø5	y 121
LF 1Ø LINE FEED	SUB 26	* 42	: 58	J 74	Z 9Ø	j 1Ø6	z 122
VT 11	ESC 27	+ 43	; 59	K 75	[91	k 1Ø7	{ 123
FF 12	FS 28	, 44	< 6Ø	L 76	\ 92	l 1Ø8	¦ 124
CR 13 RETURN	GS 29	- 45	= 61	M 77	] 93	m 1Ø9	} 125
SO 14	RS 3Ø	. 46	> 62	N 78	^ 94	n 11Ø	~ 126
SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	127 RUBOUT (DEL)

X

APL★PLUS®/2000

APL PRODUCTIVITY IN
THE VAX® ENVIRONMENT

"Of the available approaches to increasing productivity—more capital, innovation, a better trained or motivated workforce—innovation clearly offers the best return on investment."

International Data Corporation in "Computer Systems and Services for Business and Industry"

These days, increasing productivity is a top priority within business and industry. That's fine with us. We've grown from a one-office location to an international corporation in less than ten years because we specialize in a productivity enhancing service. And we've built upon what we sell.

APL (A Programming Language) is the computer programming language in which STSC specializes. It could just as easily be called A Productivity Language. As the largest supplier of APL services in the United States, STSC's intensive research and development efforts are committed to enhancing and extending APL to run in a variety of environments and in the most productive, powerful manner possible.

In the past decade, our enhancements to APL, incorporated into our proprietary APL*PLUS* System, have allowed the development of superior computer applications for many business applications with only *one-fourth* the effort required in other computer languages. STSC has further extended these enhancements to run on a variety of large-scale computer systems.

APL*PLUS/2000—the APL*PLUS System for the VAX-11/780 and VAX-11/750 computers!

How the APL*PLUS/2000 Boosts Programmer Productivity

Programmers cite five key areas where their time is spent:

- application design
- documentation
- code design and coding
- debugging
- testing.

Stated simply, faster application development increases productivity. While APL does shorten the design and documentation cycle, APL's greatest strength is at the coding, debugging, and testing stages. Here the effort is one-fourth that of other languages. In general, one character in APL is equal to an entire statement in other programming languages.

"APL is one of the most concise, consistent, and powerful programming languages ever devised."

"Since computer programming typically involves a great deal of work with various kinds of data structures, the simplification offered by APL's rich and powerful handling of arrays is central to its strength."

Daniel D. McCracken, Past President of the Association for Computing Machinery (ACM)

Brevity means money. Clearly, it means productivity. The same improvement factor of four to one for writing programs in APL applies at every stage of building computer software: APL makes it easier to conceive the solution and faster to write the program, results in fewer clerical errors, requires less documentation, and accommodates change more readily. Code logic more closely follows the logic of algorithms, making debugging easier.

Imagine application development in one-fourth the time. Imagine subsequent debugging, testing, and modification in up to one-tenth the time required using other programming languages!

"According to the 1980 report ... programming personnel spent 55 percent of their time on existing program maintenance and enhancements, the latter more often meaning 'getting the bugs out.'"

Infosystems *publisher, Arnold E. Keller, discussing the results of an INPUT survey of DP managers.*

Since over half of the programming done today is in the area of maintenance, APL can have a very significant impact on the life cycle costs for major applications.

The efficiency benefits are clear. If you can write a program in one-fourth the normal time, and modify and maintain that program in up to one-tenth the time of other languages, you have given yourself some valuable performance factors:

- You can start to program a low-cost working model of an idea as soon as the initial design concept is conceived.
- APL programming can be a means of developing firm specifications. You can explore alternative approaches and throw away an inadequate design.
- Feedback you acquire from end users in the early stages of a project improves the design, assures timeliness and acceptance of the design, and facilitates training.
- Management a[illegible]es more control over the programming process instead of being told: "It's too late to make a change."

Look at these additional productivity-increasing benefits:

- APL*PLUS/2000 appeals to the end user as well as to the computer professional. Many who would not otherwise make direct use of the computer in their work find APL to be a natural problem solving tool. It will be easier to solve the problem using APL*PLUS/2000 than to describe it to someone else! Your programming time is freed to work on [illegible]ve complex projects.
- APL*PLUS/2000 [illegible]ven. Its features have be[illegible]ed by some of the world's largest companies for over a decade in a commercial time-sharing setting—the most stringent testing environment for any pr[illegible]

The APL*PLUS/2000 Package

The APL Language Processor. Integral to the APL*PLUS/2000, the APL Language Processor provides functional compatibility with STSC's enhanced APL, *plus* additional capabilities such as a full-screen editor for quick and convenient editing of functions and data. Some of the productivity-boosting features provided are described below.

- The APL*PLUS/2000 offers a wide range of **system functions and variables** for control of the application environment and efficient use of virtual workspaces, including capabilities such as:
  - —*⎕LOAD* and *⎕QLOAD* Overlays the active workspace with a copy of a saved workspace. These functions are extremely useful in large applications that use overlay and chaining techniques.

[illegible] Allows function [illegible]ation and definition [illegible] program control.
—[illegible] Performs a string search operation on character as well as numeric arguments, locating all occurrences of a given string (vector) in another string (vector).

- STSC's ⎕FMT **formatting function**—the most powerful formatter in APL—combines the best of FORTRAN formatting notation and COBOL picture editing. Features of ⎕FMT that make it such a valuable tool for detailed tabular reporting requirements are listed below.
  —Several data arguments can be formatted at the same time.
  —[illegible] can be formatted [illegible] decorations and [illegible] effects, especially [illegible] for reporting in a [illegible] analysis environment. For example, special [illegible] may include [illegible] dollar signs and enclosure of negative quantities in parentheses.
  —Both relative and absolute tabbing are supported.
  —The printing order of data can be varied from the standard left-to-right order.
  —Message text can be inserted in the report.
  —Picture formatting allows the user to arbitrarily mix text and data in the formatted output.
- The **Exception Handling facility** automates and expands an APL program's ability to react to errors and exceptions that can occur during program execution. Exception handling, or "error trapping", can be used to increase the reliability and efficiency of APL programs by allowing the system to react to errors in a controlled manner. It can also improve the security of applications, protecting the applications, and the data they contain, from interference. In short, exception handling can be used to write applications that are tolerant of users who have minimal training on the system.
- The APL★PLUS/2000 provides access to two **file systems.** You can access your existing VMS/RMS native files. You are also provided with a component-based file system enabling you to store, retrieve, share, and update large volumes of data. Each component is an entire APL data array—not just a record. STSC pioneered this easy-to-use file system in 1970. Since then, more than 2,000 companies have made use of the file system's flexibility and reliability.

User-Oriented Utilities. During the course of 11 years as a commercial time sharing supplier, STSC has developed numerous APL utilities that aid in the readability and maintainability of source code and increase the productivity of your staff in meeting application program requirements. These utilities include:

- Program debug and documentation aids that provide a simple and efficient method for workspace documentation, and programs that search specified functions in order to aid in debugging and editing.
- Production subroutines that accept and manipulate input and grade, rank, and sort arrays of data.
- Online computer-aided instruction that teaches how to write and use APL programs. Topics include use of stored programs and inserting and modifying instructions.
- News "bulletin board" system that allows the posting of headlines and variable-length, memo-type messages for the entire APL user group.

Complete Documentation. Three separate manuals describe formatting, files, and system features. A supply of each is provided with the APL★PLUS/2000 product. Also offered with the APL★PLUS/2000 are two reference books coauthored by STSC personnel and widely read in the APL community: *APL: An Interactive Approach* (Wiley, 1976) and *APL in Practice* (Wiley, 1980).

Training. STSC provides initial user training with the APL★PLUS/2000, conducted by experienced STSC software specialists. STSC also holds regularly scheduled classes and seminars. Our wide range of topics is designed to give you the latest information and to help you get top performance from our products. APL training is an STSC specialty.

Ongoing Support for APL★PLUS/2000 Users

Experienced professionals are available to translate APL★PLUS/2000 capabilities into practical solutions for management problems. STSC can provide program development consulting to help design or complete a major application.

For day-to-day information needs, STSC provides a toll-free "hot-line" service to our software specialists.

APL★PLUS/2000 is a Product You Can Grow With

The technology behind the APL★PLUS/2000 language processor assures you of low-cost portability from one computer architecture to another. APL★PLUS system and application programs are easily transportable—whenever your overall corporate needs dictate a move to different hardware. This means you can migrate to even an inhouse mainframe computer without rendering your applications software obsolete. Count on STSC to be ahead of state-of-the-art in language development.

Future developments for better efficiencies using APL include an APL compiler, extensions to the APL language, a relational database management system, and additional application development tools.

APL★PLUS/2000 provides the core foundation to use other STSC products written in APL that deal specifically in the areas of general ledger, long range planning, financial reporting, budgeting, corporate modeling, and FASB compliance reporting.

STSC: The Company

STSC, Inc. specializes in financial management services and provides remote access computing services and software products based on the APL computer programming language. STSC was founded in 1969 as a computer time sharing firm specializing in the use of APL. We developed and promoted many of the advanced features that are viewed today as APL standards.

As the leader in APL services, STSC provides a full range of support and consulting services at each of its 25 offices in the United States and Europe.

stsc

Corporate Headquarters

STSC, Inc.
7316 Wisconsin Avenue
Bethesda, Maryland 20014
(301) 657-8220

International Headquarters

APL•PLUS International
747 Third Avenue
New York, New York 10017
(212) 751-9305
TWX 7105812254

Marketing Offices United States

Boston
462 Boylston Street, Suite 305
Boston, Massachusetts 02116
(617) 267-6864

Manufacturing Services
462 Boylston Street, Suite 305
Boston, Massachusetts 02116
(617) 267-6864

Chicago
1550 Spring Road
Oak Brook, Illinois 60521
(312) 530-7600

Dallas/Ft. Worth
1525 Elm Street, Suite 2660
Dallas, Texas 75201
(214) 263-4577

Denver
42 Denver Technological Center
7965 East Prentice Avenue
Englewood, Colorado 80111
(303) 779-8878

Detroit
100 Renaissance Center
Suite 2910
Detroit, Michigan 48243
(313) 259-0220

Hartford
111 Pearl Street, Suite 401
Hartford, Connecticut 06103
(203) 549-0107

Honolulu
1269 Maleko Street
Kailua, Hawaii 96734
(808) 261-3751

Houston
11 Greenway Plaza, Suite 2116
Houston, Texas 77046
(713) 850-9400

Los Angeles
2900 31st Street, Suite 110
Santa Monica, California 90405
(213) 450-4611

The Optimation Group
21243 Ventura Blvd., Suite 240
Woodland Hills, California 91364
(213) 340-4611

New York
747 Third Avenue
New York, New York 10017
(212) 751-9305

Philadelphia
The Bourse Bldg., Suite 530
Independence Mall East
Philadelphia, Pennsylvania 19106
(215) 627-5300

Rochester
3000 Winton Road South
Townline Park Building E
Rochester, New York 14623
(716) 442-5281

San Francisco
Spear Street Tower
1 Market Plaza Building
Suite 1601
San Francisco, California 94105
(415) 777-4357

Maintenance Systems Group
Spear Street Tower
1 Market Plaza Building
Suite 1601
San Francisco, California 94105
(415) 777-4357

Southeast Marketing
400 Eastowne Drive, Suite 108
Chapel Hill, North Carolina 27514
(919) 493-2478

Washington, D.C.
7101 Wisconsin Avenue
Suite 1414
Bethesda, Maryland 20014
(301) 986-1750

Westchester-Fairfield
11 Clearbrook Road
Elmsford, New York 10523
(914) 347-5560

International Offices

United Kingdom
APL•PLUS Limited
50-52 Chancery Lane
London WC2A 1HL, England
01-242 8135

Spain
APL Informática S.A.
(Independent Distributor)
Rosario Pino 6
Madrid-20, Spain
(1) 279-47-84

France
Société de Traitements et de
Services Conversationnels
Tour Neptune, Cedex N°20
92086 Paris La Défense, France
773-7964

Subsidiaries

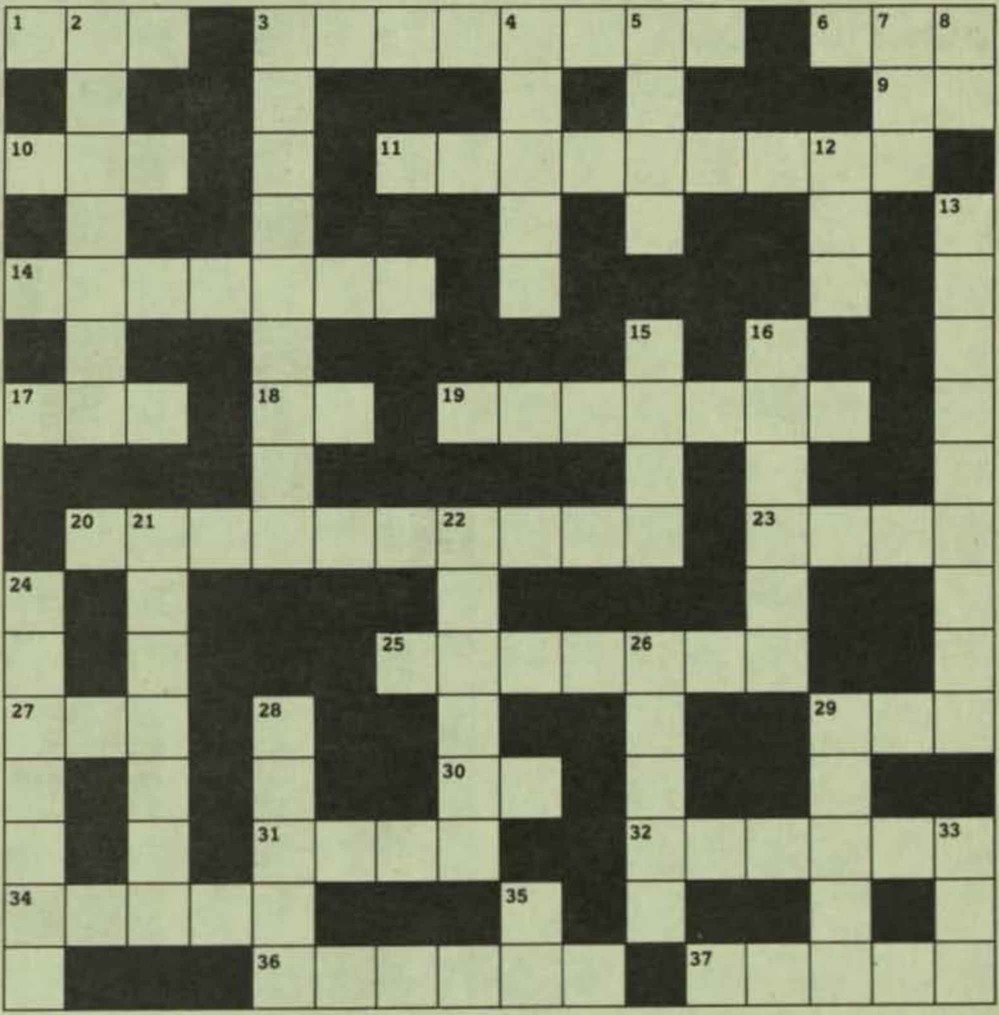
MTSC, Inc.
4330 East-West Hwy., Suite 1111
Bethesda, Maryland 20014
(301) 951-4200

Resource Systems, Inc.
42 Denver Technological Center
7965 East Prentice Avenue
Englewood, Colorado 80111
(303) 779-6909

Printed in the U.S.A.
CC057-81

stsc

APL A PuzzLe



SEE REVERSE SIDE FOR SOLUTION

ACROSS

1. SIMPLIFY (ZA>0)-ZA<0
3. RETURN THE AVERAGE OF A NUMERIC VECTOR *V*
6. RETURN TWICE THE VALUE OF *M*
9. SIMPLIFY ⍴*M*>1
10. GENERATE THE INTEGERS FROM 1 TO THE POSITIVE SCALAR INTEGER ZA
11. RETURN THE INTERSECTION (ELEMENTS COMMON TO BOTH) OF THE TWO VECTORS *BA* AND *Z*
14. SIMPLIFY Z[(Z<B)/⍳⍴Z]
17. ASSUME A←2:   10×(A*4)+1+3×A
18. SIMPLIFY V-1|V
19. GENERATE 13 15 17 19 21 23 25
20. RETURN VECTOR *V* AS A ONE-COLUMN MATRIX
23. BYE-BYE
25. ASSUME V←⍳5:   GENERATE 9 16 25 36 49
27. ASSUME *C* IS A MATRIX: SIMPLIFY C,(1⍴⍴C)⍴0
29. FOR AN INTEGER ARRAY *F*, RETURN 1 WHERE *F* IS ODD, 0 WHERE *F* IS EVEN
30. THE APL EQUIVALENT OF A ONE DIMENSIONAL EMPTY SHOPPING CART
31. GENERATE 9 RANDOM NUMBERS FROM 1 TO 8
32. SIMPLIFY I[(I<11)/⍳⍴I]←11
34. RETURN THE NUMBER OF 5'S IN EACH COLUMN OF A NUMERIC MATRIX *Z*
36. SIMPLIFY 2×(¯1↑∧\V1)[1]
37. RETURN THE SMALLEST ELEMENT OF TWO NONEMPTY NUMERIC VECTORS *V* AND *W*

DOWN

2. REPLACE ALL 3'S IN *Z* BY 2
3. RETURN THE RANGE (DIFFERENCE BETWEEN THE LARGEST AND SMALLEST VALUES) OF A NUMERIC VECTOR *V*
4. GENERATE A LIST OF THE NAMES OF THE VARIABLES IN YOUR ACTIVE WORKSPACE
5. RETURN THE RANK OF *ZA*
7. ASSUME *A* IS A VECTOR: RETURN 1 IF *A* IS NONEMPTY, 0 IF IT IS EMPTY
8. SIMPLIFY ⌽⌽MM
12. RETURN THE POSITION OF THE FIRST OCCURRENCE OF SCALAR *S* IN VECTOR *B*
13. RETURN THE NONBLANK ELEMENTS OF A CHARACTER VECTOR *F*
15. RETURN THE LAST TWO ELEMENTS OF A VECTOR *V*
16. GENERATE THE NUMBERS FROM 1 TO 36 WHICH ARE PERFECT SQUARES (A PERFECT SQUARE IS AN INTEGER SQUARED)
21. RETURN THE POSITIVE NUMBERS IN A VECTOR *Z*
22. GENERATE THE ODD NUMBERS FROM 3 TO 17
24. ASSUME C←⍳10:   ASSIGN THE ADDITION TABLE FOR THE FIRST 10 POSITIVE INTEGERS TO THE VARIABLE NAME *T*
26. WHERE AM I?
28. SHUFFLE ⍳52
29. SIMPLIFY 2+1⍴V[⍒V]
33. RETURN THE FIRST ELEMENT OF A NONEMPTY MATRIX *W*
35. SIMPLIFY V××V

NOTE: ASSUME ⎕IO←1
⎕CT←0

SOLUTION

1 ×	2 Z	A	■	3 (	+	/	V	4)	÷	5 ρ	V	■	6 2	7 ×	8 M
■	←	■	■	⌈	■	■	■	V	■	ρ	■	■	■	9 ρ	M
10 ⍳	Z	A	■	/	■	11 (	B	A	∊	Z	)	/	12 B	A	■
■	-	■	■	V	■	■	■	R	■	A	■	■	⍳	■	13 (
14 (	Z	<	B	)	/	Z	■	S	■	■	■	■	S	■	'
■	=	■	■	-	■	■	■	■	■	15 ¯	■	16 (	■	■	
17 2	3	0	■	18 ⌊	V	■	19 1	1	+	2	×	⍳	7	■	'
■	■	■	■	/	■	■	■	■	■	↑	■	6	■	■	≠
■	20 (	21 (	ρ	V	)	.	22 1	)	ρ	V	■	23)	O	F	F
24 T	■	Z	■	■	■	■	+	■	■	■	■	*	■	■	)
←	■	>	■	■	■	25 (	2	+	V	26)	*	2	■	■	/
27 C	.	0	■	28 5	■	■	×	■	■	W	■	■	29 2	|	F
∘	■	)	■	2	■	■	30 ⍳	0	■	S	■	■	+	■	■
.	■	/	■	31 ?	9	ρ	8	■	■	32 I	←	I	⌈	1	33 1
34 +	⌿	Z	=	5	■	■	■	35 |	■	D	■	■	/	■	ρ
C	■	■	■	36 2	×	∧	/	V	1	■	37 ⌊	/	V	.	W

STSC, Inc.
2115 East Jefferson Street
Rockville, Maryland 20852
(301) 984-5000

APL.68000

Microcomputer Interpreter

THE COMPUTER COMPANY • MICRO APL SYSTEMS
1905 Westmoreland Street, Richmond, VA 23230-3297 • 804/358-2171

APL.68000 Microcomputer Interpreter

The Computer Company's APL.68000 is the first APL interpreter for the M68000 microcomputer chip which offers state-of-the-art processing speed, and up to 16 megabytes of directly addressable main memory, limited only by available hardware. APL.68000 is a full implementation of IBM's APL.SV, with the same enhancements as our commercial timesharing system ACTION/APL.SV.

If you have used APL, you know that its strengths as a programming language—concise code, easy debugging and less detailed specification work, for example—mean increased programmer efficiency, reduced costs and fewer programming errors. And you know that APL is incomparably versatile in providing new solutions for difficult applications—quickly.

If you are familiar with hardware development, you know that microprocessors are the new generation in computing. Their small size, low cost and increasingly impressive main storage make them, for many businesses, an irresistable bargain.

With APL.68000, you can have both. The distinct advantages of a microcomputer based on the M68000 chip and a sophisticated APL interpreter proven for more than 12 years by our many timesharing customers.

Consider the special features of APL.68000:

POWERFUL FILE SYSTEM

- User files are always open; there is no need to open (tie) or close (untie) them.
- Primitives, ⍈ ⍇ ⍐ ⍗, rather than shared variables, are used for file operations.
- Files may have an arbitrary number of components.
- Each component may contain data of an arbitrary size, shape and type.
- Components may be inserted between existing components, deleted, replaced, or appended to the front or back of the file.
- Files may be protected by passwords.
- Files have an access matrix to specify how a user may access the file. A user may be allowed to append components to a file, for example, but not allowed to insert or replace components.
- Files may be renamed or replaced.
- You can access dynamically a list of all of your files, as well as information about each file and each component.
- A file is created automatically by the first file write.

OVERLAYS (PACKAGES)

- Through the use of overlays or packages, functions and variables may be clustered and written to a file for subsequent quick and easy access.

COMMERCIAL ALPHA FORMATTER

- The alpha formatter uses a COBOL-like picture statement, allowing you flexibility in inserting text such as dollar signs, commas and slashes between output digits. For example, ¯146928 could be formatted to print as $1,469.28 *CR* .
- You can work with both fixed and floating fields, a feature especially convenient in formatting financial statements. For example, you can specify that dollar signs be placed automatically next to the highest digit.
- You can define the fill character for a field, a feature useful in writing checks or in any application using a special fill character.
- The fixed or floating sign minus -, high minus ¯, parenthesis (, and brackets [,], will be printed only if the number you format is negative. The plus sign + will be printed only if the number you format is positive. All other fixed or floating characters are not affected by sign.
- Text following the last digit field in the picture statement is displayed only if the number you format is negative. This feature allows you to include text such as *CR* following negative dollar amounts.
- You can suppress all leading zeros or force significance in your output. For example,

```
      'ZZZ'⍺2
  2
      '999'⍺2
002
```

APL.68000 Benchmarks
In Milliseconds

Benchmark		Microcomputers			Minicomputers		
		APL.68000 8 MHZ 1 Wait State	IBM 5120	APL/V80	HP 3000 Series 11	HARRIS S123	DECSYSTEM 2020
Plus Reduction	Z←+/VI	121.0	141.3	128.3	50.6	1.9	3.5
Logical Reduction	Z←∨/VL	4.4	57.9	5.8	5.4	0.7	6.8
Maximum Reduction	Z←⌈/[1]MI	52.7	136.2	95.3	39.3	2.3	5.3
Exponentiation	Z←VI*.5	6603.6	4743.1	39754.7	654.7	57.2	103.9
Absolute Value	Z←|VR	99.6	134.0	109.7	101.7	3.6	8.1
Indexing	Z←VR[VI[⍳20]]	33.5	184.3	77.8	34.4	5.3	5.4
Sorting	Z←VI[⍋VI]	590.5	3539.6	2375.3	288.1	52.9	56.0
Take	Z←¯2 1↑MR	9.3	53.2	591.5	5.1	1.0	2.8
Membership	Z←VI∊VI	210.3	38036.6	7696.3	3871.0	38.9	504.3
Transposition	Z←2 1⍉MC	176.6	1748.7	1928.3	11.9	138.5	82.1
Outer Product, Characters	Z←VC∘.=VC	243.4	5353.7	291.3	306.9	21.1	35.8
Outer Product, Integers	Z←(⍳50)∘.+⍳50	339.1	6585.4	1850.8	1193.8	53.9	54.6
Inner Product, Real Numbers	Z←VR⌊.+VR	199.0	737.3	2646.8	95.3	7.0	12.3
Matrix Division	Z←MR⌹10↑VR	495.7	1350.8	10436.7	104.3	10.0	33.6
Fibonacci Series	Z←1 1 L:→(100>⍴Z←Z,+/¯2↑Z)/L	4211.0	10770.0	6206.6	2373.4	349.4	1534.6

Note 1: Variables used in the benchmarks.

```
MI←10 10⍴VI←(500⍴0 1 0 0 1)/⍳500
VL←1 0 1 1 0 0 0 1
MR←10 10⍴VR←VI+0.1
MC←26 26⍴VC←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

Note 2: The DECSYSTEM 2020 used APL-SF. The HP 3000 used APL/3000. VSAPL was used on the IBM 5120.

Note 3: All times are in milliseconds.

For additional information, please contact
Phil Van Cleave at 804/744-2458.

THE COMPUTER COMPANY • MICRO APL SYSTEMS
1905 Westmoreland Street, Richmond, VA 23230-3297 • 804/358-2171

APL.68000 Retail Price List

Effective October 1, 1981

Microcomputer Interpreter	$ 3000.00
User Manual, if purchased separately	30.00
Reference Card	*No charge*
Benchmarks Report	*No charge*
Product Brochure	*No charge*

Stated prices are retail prices for single end-users; manufacturer, OEM, and multiple end-user discounts are available.

In the United States, APL.68000 is currently implemented on WICAT Systems hardware, under the DIS operating system, and on Motorola hardware under the VERSAdos operating system.

In Europe, APL.68000 is available on MicroAPL LTD SPECTRUM hardware, under the MIRAGE operating system.

To place orders, or for additional information,
please contact Phil Van Cleave at 804/744-2458.

THE COMPUTER COMPANY • MICRO APL SYSTEMS
1905 Westmoreland Street, Richmond, VA 23230-3297 • 804/358-2171

Using the special features of the alpha formatting primitive, you can produce descriptive output.

```
      '((Z,ZZ9.99)'⍺1469.28   0  ¯346
1,469.28         0.00     (346.00)
```

FAST SEARCH/REPLACE PRIMITIVE

APL.68000's search/replace primitive is extraordinarily fast. It will allow you to perform character vector search to locate and/or replace parts of your text. Any part of the text can be stored as a variable to facilitate your corrections. For example,

```
        A←'MARY HAD A LITTLE LAMB'
        ⎕SS(A;'LITTLE')
12 (LITTLE begins at position 12 of the string)

        ⎕SS(A;'LITTLE';'BIG')
MARY HAD A BIG LAMB
```

DELIMITED BLANK REMOVAL PRIMITIVE

⎕DBR removes all leading, trailing and duplicate blanks from a character vector and all blanks preceding and following any set of specified delimiters. For example,

```
        ', '⎕DBR'   L1 ,  L2   L3  '
L1,L2 L3
```

DELIMITED VECTOR-TO-MATRIX AND MATRIX-TO-VECTOR PRIMITIVE

Using ⎕BOX , you can define your delimiter for rapid conversion of vector to matrix or matrix to vector. This is especially useful in preparation of row and column headings. For example, using the semicolon ; as the delimiter,

```
        ';'⎕BOX 'DOE, JANE;LITTLE, TOM'
DOE, JANE
LITTLE, TOM
```

COMMENTS

You can add a comment following an APL expression. For example,

```
        2×5 ⍝ MULTIPLY 2 TIMES 5
```

DIAMOND STATEMENT SEPARATOR

The diamond ⋄ inserted between statements allows you to place multiple statements on one line. For example,

```
      2+6⋄A←1 2 3⋄3×A
8
3 6 9

      A
1 2 3
```

ERROR TRAPPING

Error trapping in APL.68000 gives you the option of intercepting and handling errors encountered during the execution of a function. ⎕ERX sets the error trap and ⎕LER returns the program line number and type of error found.

EXECUTION OF SYSTEM COMMANDS

The Execute primitive allows execution of system commands and single-line function definition statements. For example,

```
      ⍎')LOAD DEMO'
SAVED 12.10.06 06/10/81
```

Execution of function definition statements:

```
⍎'∇STAT[2]R←+/X∇'
```

SHARED VARIABLE PROCESSORS

One processor allows APL to send and receive data through any I/O port on the system; the other processor allows you to read and write standard operating system files.

AUTOMATIC SYMBOL TABLE CLEANUP

Symbol tables in APL.SV often become cluttered with unused names, returning SYMBOL TABLE FULL errors. APL.68000 automatically removes unused names from the symbol table.

The Computer Company
1905 Westmoreland Street, Richmond, Virginia 23230-3297 • 804/358-2171

Houston
The Computer Company
5301 Hollister, Suite 430
Houston, Texas 77040
713/460-5955

Los Angeles
The Virginia Computer Company
77 Brookhollow Drive
Santa Ana, California 92705
714/754-6440
213/678-9605

New York
The Computer Company
211 East 43rd Street, Suite 1804
New York, New York 10017
212/682-7390

Richmond
The Computer Company
1905 Westmoreland Street
Richmond, Virginia 23230
804/358-2171

October 1981

10268583 8