Burroughs

B 6700/B 7700

APL/700

USER REFERENCE MANUAL

Burroughs Corporation
Detroit, Michigan 48232

$3.00

Printed in U.S. America **1 April 1974** **5000813**

Burroughs Corporation believes the program described in this manual to be accurate and reliable, and much care has been taken in its preparation. However, the Corporation cannot accept any responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

Table of Contents

Table of Contents (cont)

Table of Contents (cont)

List of Illustrations

List of Tables

INTRODUCTION

This publication is a user reference manual for the Burroughs APL/700 System, which is a general-purpose system for processing information in a time-sharing, interactive environment. APL (A Programming Language) is a language for describing procedures in the processing of information and is very concise, consistent, and powerful. Because APL is similar in many respects to algebraic notation and contains many useful functions not expressible concisely with conventional symbols, it is very effective for describing algorithms (problem-solving procedures). The Burroughs APL/700 System incorporates all of the features of existing APL systems, but it has many additional features and enhancements that make it more versatile and powerful. Some of the salient features and enhancements are:

- Extended function capability
- Improved interaction
- Formatting capability
- Extended function editing
- Enhanced system control
- Comprehensive error reporting

The purpose of this manual is to provide sufficient reference data and instructions to assist users in the application and use of the Burroughs APL/700 System. The intent of the presentation is to enable the user to take best advantage of the APL/700 capabilities and to achieve maximum effectiveness in his application.

SECTION 1

APL/700 SYSTEM DESCRIPTION

GENERAL.

APL/700 is an interactive tool for problem solvers. Its purpose is to provide a means for the person formulating the problem to get results quickly. The user works through a terminal. Entry of problem formulation and data can be intermixed. Entered information and returned results are displayed for immediate review. This type of use is appropriate for applications where user contribution to solution is important. Some characteristics of effective uses include value in timely solution, experimentation, asking what if questions. These contrast with traditional bulk data processing, where massive outputs are prepared in hope that somewhere therein can be extracted the answers to any potential questions.

The problem formulation can often be in terms of an immediately executed APL expression for which direct response is provided. APL/700 has many powerful primitive functions built-in and available for this use. These apply consistently to simple or structured data. Uniform, parallel processing of all elements in a data structure permits significant processes to be concisely expressed, with irrelevant detail suppressed.

Entered data or the results of computations can be retained in variables. User defined functions can be used to capture formulated problem solutions. User defined functions are composed from constants, variables, the primitive functions, and other user defined functions. Captured functions and data can be saved for subsequent use. A file system is available to extend the amount of data available for retention, retrieval and processing.

APL is being successfully used for many applications including:

- Financial analysis and forcasting
- Statistical analysis
- Administrative reporting
- Mathematical analysis
- Text processing
- Mailbox message distribution
- Reservation control systems
- Computer aided instruction
- Graphing and data plotting
- Simulation

The common property of these applications is their use of direct input and immediate display response. Traditional computation-bound applications are not candidates for APL, at least in that form. Often recasting these traditional jobs into APL provides a more satisfactory solution for the user with the problem.

PROPERTIES AND FEATURES.

APL/700 may be characterized as:

accessible	immediate response for "trivial" requests
inobtrusive	problems quickly solved at user's pace
concise	powerful primitives on data structures
simple	consistent, few rules
readable	define functions in few lines
forgiving	easy error correction, good recovery
secure	protection for private or shared work
habit forming	accomplishments eagerly shared among users

Features that make APL/700 an effective interactive system include:

- built-in APL functions for processing data
- immediate expression execution in calculator mode
- progressive expression development by augmenting prior entry
- data entry in calculator or input modes
- user function creation in function definition and editing mode
- file operators for accessing extensive data
- formatting functions for report preparation
- system functions and commands to query and alter environment
- keyboard input and display controls

USE REQUIREMENTS.

The requirements to use APL are simple:

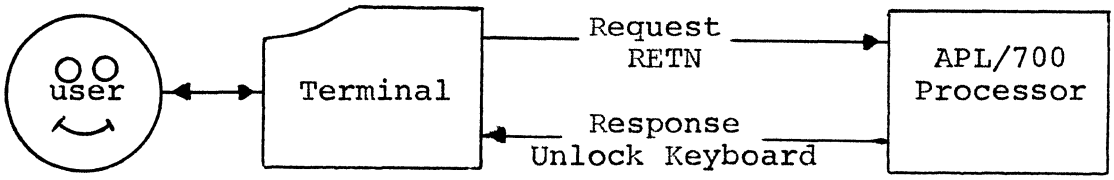
- a terminal with APL characters
- an account on an APL/700 system

The above are presumed available. Note that typing skill is not on the above list. APL is so concise that typing is not a significant barrier for most problem statements. Since the reader is encouraged to learn APL on a terminal, keyboard familiarity develops with use. The user quickly learn to experiment: When in doubt, try it!

APL/700 INTERACTIVE ENVIRONMENT.

The user of APL/700 apparently receives access to a central processor always ready for his exclusive use. This is achieved for many users concurrently, since the amount of computer resources required for servicing any one user is usually a small fraction of that available.

A transaction is the alternating cycle starting with a user phase followed by a processor phase. The user phase starts with the terminal ready for user typing, continues through user typing an entry requiring service and completes with typing return (RETN). The processor phase starts by receiving the RETN, continues with servicing the request, possibly generating output, and finally restoring the terminal ready for next user entry.



Phase	User	Processor
	Types Entry RETN	Process (Output)
Keyboard	Unlocked	Locked
Typical Time Span, Seconds	1 to 30	0 to 1 0 to 6

FIGURE 1-1. TRANSACTION CYCLE

So long as the time in the user phase or receiving output is essentially all the transaction cycle, the user is satisfied that he has the entire processor. It is always waiting for him, and he works at his own pace. APL/700 achieves this by scheduling "short" requests (taking no more than a fraction of a second of processing to complete) for almost immediate service. "Involved" requests (that a user might expect to take a while) are scheduled for processing that can be interrupted as necessary to service short requests from other users.

The benefits from sharing the APL processor among many users concurrently include:

Immediate response for short transactions,
Work smoothing among many irregular demands for service,
Powerful processor available when needed,
Cost spread across users as resources are used,
"Think time" need not be penalized,
Data files for data accumulation and shared access.

DATA ELEMENTS AND OBJECTS.

Data objects are the units for processing. A data object has the properties of type, shape, and value.

The type of a data object is one of the following:

character	any APL characters
numeric	any value representable as a number

The numeric data type is further restricted to integer, having integer values only; and Boolean, having values 0 or 1 only.

A data object may be a scalar, in which it has a single element without shape (a geometric point). A data object may be an array, in which case it has a number of dimensions, with a length or number of elements along each dimension. If there is only one dimension, the array is referred to as a vector. The shape of a data object is the vector indicating the lengths along the dimensions. The right most element of the shape vector is the number of columns. A two dimensional array is referred to as a matrix. The shape of a matrix is the number of rows, then the number of columns. Any dimension may have length 0 or 1. The result of processing such objects requires special attention. The rank of an object is the number of dimensions.

The size of an object is the number of elements it contains. The size is empty if the length of any dimension is zero. The size of a scalar is one, as is the size of a one element array of any rank. A character string is a scalar or vector of character type.

The value of a data object is the array of values of the individual elements.

Any array data object may be characterized as:

rank n	n dimensions
rectangular	all planes across a dimension have the same shape
homogeneous	all elements are the same type
dense	all elements are present, and represented the same way

The geometric view of an array is often useful. A plane is a slice of a shaped object that is orthogonal to a given dimension of that object. A plane across the k-th dimension of an n-dimensional object is a (n-1)-dimensional object with all but the k-th dimension of the original retained. Thus a plane of a vector is a scalar element, and a plane of a matrix is a row or column. A vector along a dimension is parallel to that dimension's axis. A plane across a dimension is orthogonal (at 'right angles') to that dimension.

A face of an array is the first or the last plane across any dimension. A corner of an array is any n-dimensional sub-array having n of its faces that are sub-faces of an n-dimensional array.

Each element in an array can be referenced by its coordinates, an ordered list of scalar indices, one for each dimension from first to last. An index refers to the ordinal position along a dimension of a plane across that dimension. The index position indicates the position of a plane relative to a dimension of an array. Index position j of a plane selects the j-th plane along a given dimension. The index domain for a coordinate is the set of integers starting from the origin and including one member for each plane across that dimension. For any dimension of an array, any numeric data object can be used that contains only integers in the index domain for that dimension. A semicolon separates index objects referring to successive dimensions. Any dimension without explicit index objects is equivalent to all planes across that dimension. The index sequence is the conventional order of the sets of indices of the elements of an array, whereby the last dimension steps through its values most rapidly. (As [1;1] [1;2] [2;1] [2;2]).

USAGE MODES.

The user of APL may select one of the following modes for use at any time.

Calculator Mode.

immediate execution of entered expressions

progressive expression development

assignment of values to variables

calling defined functions

prompt: indent five spaces

Function Execution Mode.

execution of user defined functions

prompt: keyboard locked

Data Entry Mode.

numeric, in response to default prompt □:

character, in response to optional user generated prompt

Function Definition and Editing Mode.

development of user defined functions

convenient line editing

establishment of automatic debugging aids

prompt: [n] at left margin for line n of open function

COMPONENTS OF APL LANGUAGE.

The APL user makes use of the following kinds of components:

constants and variables
primitive functions
user defined functions
control structures

CONSTANTS AND VARIABLES.

A constant is a data object without a name. Constants can appear as part of user defined functions or entered as part of calculator mode expressions.

A variable has a name that gains its meaning from assignment of a data structure. The variable name is used in APL expressions as reference for the associated data structure. Each subsequent assignment to the same name changes the data structure associated with it.

Constants and variables can be used as arguments to functions in APL expressions.

FUNCTIONS.

Functions perform processing following particular defined rules. Many primitive functions of general utility are built-in to APL. Other functions can be created by the user for his problem solving. These are called defined functions. They make use of primitive functions or other defined functions.

A function accepts arguments and generally returns a value, resulting from following the processing rule for that function as applied to the argument values.

A function is defined for a domain of values for each of its arguments and produces a result in the range of values. For example the relational function "less than", as used in

A "less than" B

has numeric domain for arguments A and B and the values true and false as the range of values for the result.

In APL, "less than" is expressed by the symbol '<', and the values true and false by the Booleans 1 and 0 respectively.

Primitive Functions and Operators.

Complete families of primitive functions are provided for numeric type data objects:

- arithmetic
- relations
- logical
- higher functions
- random numbers

Additional function families exist that apply to both numeric or character data types:

- structure building and changing
- mixed type
- sets
- indexing
- assignment
- formatting

A group of operators exist which use primitive functions as compcnents that operate with data objects.

A file system provides convenient access to extensive data using a set of file operators.

A set of system functions permit querying and altering the environment within which APL is used.

A similar set of system commands can be used only in calculator mode.

User Defined Functions.

User defined functions may be created by combining built-in primitive functions, operators, constants, variables, other defined functions, punctuation, and control structures to perform more complex processing than can be done by single primitive functions.

A user defined function can have arguments. Arguments provide values for use during execution.

A user defined function may return a result from execution. If so, the user defined function can be used similarly to primitive functions to compose expressions.

CONTROL STRUCTURES.

The APL control structures control order of execution. Primitive functions apply "in parallel" to all elements of the structures. Functions are elaborated right to left within expressions. Lines are normally executed in sequence within user defined functions. Non-sequential execution is achieved by explicit transfer to a line number (which may be computed). A user-defined functions may be called, in which case control is passed to the called function. Subsequently control is returned to the caller after the point of call. The completed function may return a value. Functions may be called recursively.

There are no formal conditional or iterative control structures for user defined functions. These are generally subsumed within processing on data structures. When required, the conditional and iterative control structures are synthesized by explicit control transfers.

WORKSPACES, LIBRARY AND FILES.

Each user account has an active workspace. The active workspace is the fixed size area of storage in which a user conducts his transactions. At first sign-on this workspace is unnamed and clear. After some transactions, there may be some variables having values and some user defined functions having continuing use.

A user can name the active workspace and save a copy of it in his library. Library workspaces are inactive, in that they are not currently being worked upon. A library workspace can be reactivated. The number of workspaces in the user library is limited to the quota established by the installation for that account. All workspaces have the same size, determined by the installation.

Within a workspace are all retained variables, defined functions, and temporary storage required during processing. The conciseness of APL/700 defined functions permits a surprisingly large processing capability in a workspace.

Each account may also have a quota of files. Each file has a name and a set of numbered components. Each component is either null (having no content) or contains an APL/700 data object. Data objects can readily be exchanged with the active workspace. This increases the amount of data that can be processed by functions in a workspace.

SELF PROTECTION.

The active workspace contains current work. Whenever desired, a copy of that workspace can be saved for subsequent resumption at the point of saving.

Function definition changes or experimental computation can be done, then either kept if good, or discarded by returning to the formerly saved version.

The active workspace is retained in the event of unexpected disconnection caused by either the terminal, the communications link, or the main system. Upon next sign-on for the account, recovery occurs automatically to within the last (incompletely) entered transaction if in entry phase; or to the last line processed if in processor phase.

The commands having irrecoverable effects tend to be separated and protected against accidental misuse. For example, the user can ERASE names of variables, functions or groups; but must DROP a workspace.

SECURITY AND SHARING.

Protecting an account, its workspaces and files from other users is often important to a user. Locks and passwords provide these capabilities. Selective sharing of workspaces and files among accounts is often desirable. A user can grant access privileges to those he wishes, and deny privileges to all others.

A function can be locked so that it cannot be opened for examination by any workspace other than where it was locked.

A user account name is unique, assigned by the installation. It is not considered private, but only a means for identifying the account when signed on the system, and for other users to reference the inactive workspaces and files retained for it.

The account user can add a distinct password to any account, workspace or file name. Password use can provide a degree of security to the name there locked, since the assigner of that password controls its dissemination. The password is not publicly known within the installation. A password can be changed at any time. Passwords and changes thereto are entered through the terminal. If hard copy is printed, a blot can be used to obscure by overprinting the area in which the password appears. Of course, no security is provided against tapping the communications line connecting the terminal with the APL system unless special communications security means are taken.

A user cannot alter the original of a workspace in another account; he can only obtain a copy of it in his active workspace, assuming the account owner has told him the account name and workspace name (and password if any).

A user can alter any file in the APL file system, if he knows the owning account/file name (and password if any). Thus to control the allowable file alteration in shared file applications, the owner should provide a file access function through which all accesses are made. In this function, the file password can be secured from disclosure and necessary access conditions can be checked. This function can itself be locked, denying the user the ability to examine its content. Thus, the file name and password need never appear in visible form to the user.

A file can be shared among several users. The capability to update a file component safely is provided through reserving the file for exclusive use during the update operation.

SECTION 2

INTERACTING WITH APL/700

GENERAL.

The APL/700 environment consists of a very distinctive conversational interface, where the user intimately interacts with the system in performing transactions or generating, executing, and debugging a program. During the process of execution, the user may at any time stop the processing and have available all of the local variables and other environmental factors for examination and modification. Because the user is able to follow the processing in such detail, he has a very effective debugging tool. The APL/700 user environment consists of an available library, workspace properties (such as print width, print precision, index, origin, and workspace name), accounting information, and other parameters. The user has control of these factors and may query them. By virtue of the system interactivity, the user has a large number of tools by which he may edit his transaction upon entry or upon reentry. The order in which an entry is typed is not relevant, but the final image of that entry is the logical entry that is sent to the system.

APL/700 is heavily oriented to problem solving; thus, a user can rapidly and efficiently form programs and interact with them. Because of the simple syntax of APL/700, users other than programmers may successfully use it. The reliability engineer, for example, who knows his application well but knows little about programming, need not communicate his application to another person, who then performs the programming chore. He can directly approach the computer by means of a terminal using the APL/700 language. APL/700 reduces the problem solution turn-around time considerably. In addition to mathematical computations, APL can also be used for modeling of complex systems or for simulation, text editing, algorithm development, probability predictions, and forecasting. Moreover, APL/700 provides a vehicle for the development of application packages, such as circuit analysis programs and linear programming techniques.

ACCESSING THE SYSTEM.

The user interface with APL/700 is through a communications terminal with the special APL typeface and keyboard configuration and with provisions for direct or acoustic-coupling teleprocessing communications with the system Data Communications Processor. The APL/700 System is compatible with all standard hard-copy and video APL-type terminals.

This section describes the configuration and operation features of the typical APL-type terminal. Application reference data and detailed procedures are given for some of the general operations required, such as sign-on, sign-off, and editing procedures. The instructions presented in this section and throughout the manual assume the use of the typical APL terminal configuration (as described herein) and an acoustically coupled telephone interface with the Data Communications Processor.

APL TERMINAL KEYBOARD CONFIGURATIONS.

Figure 2-1 shows the configuration of the most commonly available APL terminal keyboard, which has 44 character/symbol keys, 9 functional keys/bars, and a power on-off switch. Recently produced APL terminals contain 47 character/symbol keys as shown in Figure 2-2.

APL CHARACTER/SYMBOL SET. The APL character/symbol set consists of the 26 uppercase alphabetic characters, numerals 0 thru 9, standard punctuation/mathematical symbols, and a number of special APL function symbols not used on standard typewriters. Also note that the conventional symbols are not in the standard typewriter keyboard locations.

NOTE

The following APL character-set are described for typical (standard) APL terminals. Character formats for other terminals vary somewhat in form. For example, all characters on some units are upright and not condensed.

Alphabetic characters are in the uppercase, italic, and condensed (higher than wide) form as follows:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A full set of underscored alphabetic characters may also be used.

Numerals are in the upright and condensed form as follows:

1 2 3 4 5 6 7 8 9 0

Function symbols are in the normal, not-condensed, and upright form (except for Greek letters) as follows:

¨ ¯ < ≤ = ≥ > ≠ ∨ ∧ - ÷ ? ⍵ ∈ ⍴ ~ ↑ ↓ ⍳ ○ * →

⍺ ⌈ ⌊ _ ∇ ∆ ∘ ' ⍞ ( ) ⊂ ⊃ ∩ ∪ ⊥ ⊤ | ; : \ / , .

Figure 2-1. APL Terminal, Typical 44-Character/Symbol Keyboard Configuration

Figure 2-2. APL Terminal, 47-Character/Symbol Keyboard Configuration

Additional operator characters and symbols are formed for use by APL/700 by overstriking the characters and symbols as follows

NOTE

The order in which characters are overstruck is not important. Overstrike "$" is not necessary on those 47-key configurations having the dollar ($) sign.

Table 2-1 shows how the special overstrike characters and symbols are formed.

FUNCTIONAL KEYS/BARS. The APL terminal keyboard has special keys and bars for the tab, shift, space, carriage-return, backspace, attention, margin-release, and clear-set functions. Some terminals also have index or line-feed keys. In addition to the power on-off switch, some terminals may be equipped with a communicate-local function switch. The "communicate" position of this switch must be used when operating in the APL/700 system.

BASIC IMPLEMENTATION OPERATIONS.

The basic operations required to implement the APL/700 involve sign-on, transaction, and sign-off procedures. Recovery and editing procedures are also performed on an as-required basis.

TYPING RULES/CONVENTIONS.

Except for different character/symbol key locations and certain special functions and rules, typing at the APL keyboard is done in the conventional manner. The following rules/conventions apply:

Function	Rule/Convention
User/System Typing	A user can type only when the keyboard is unlocked, whereas the APL/700 system locks when processing user inputs or displaying responses. Operator typing is distinguished from system typing (on hard or video copy) by indentation in most transactions. (User copy is automatically indented five spaces, by system "prompt", while system copy normally starts at the left margin.)

Table 2-1

Forming Overstrike Symbols

Overstrike Character	Formed By		Overstrike Character	Formed By	
	Overstriking	With		Overstriking	With
ß	S	/	⊡	÷	□
≠	-	/	⊘	/	□
⊀	-	\	⊠	\	□
⊖	-	o	⊙	o	□
⊘	\	o	⊡	→	□
⊛	*	o	⊡	←	□
⊕	|	o	⊳	>	□
⍋	|	Δ	⊲	<	□
⍒	|	∇	⊽	∇	□
≜	_	Δ	⊿	Δ	□
≂	~	∇	⊼	~	□
⋎	~	v	⊻	v	□
⋏	~	^	⊼	^	□
⍝	°	∩	⊽	↓	□
⍦	°	⊥	⊼	↑	□
⍑	°	T	⊙	°	□
I	⊥	T	⊟	=	□
!	.	'	⊡	'	□

Function	Rule/Convention
Return	Pressing the return (RTN or RETURN) key signals the system that a user entry is complete and ready for processing. (The carrier is returned to the left margin and the keyboard is locked.)
Entry Length	Each user entry should fit on a single type line.
Shift (upper-lower characters/symbols)	Pressing any character/symbol key normally inserts the lower character/symbol on that key into text at position of cursor. Simultaneous pressing of SHIFT key inserts the upper symbol on key. (A shift lock can also be used.)
Spacing	Momentarily pressing the space bar positions the cursor one space to right; holding space bar on some terminals causes repetitive spacing. NOTE Extra spaces will not usually affect transaction processing.
Backspacing	Pressing backspace (BACK SPACE) key positions cursor one space to left. On some terminals, repetitive backspacing is accomplished by pressing and holding backspace key.
Tabs	Pressing tab key positions cursor rightward to next tab stop (if set). In application, the tabs should be set to constant intervals (such as five characters).
Line Feed	Pressing line-feed key discards text above and to right of current position of cursor.
Attention/Interrupt (or Break)	The attention (ATTN) key provides for initiation of special processing. It's uses include: (1) correction of immediate entry error restating and adjusting of prior entry, and (2) execution suspension or abortion.
Text Comment (Lamp) Symbol (@)	If it is desired to enter notations or comments without system response (niladic), prefix the comment text with the lamp symbol.

Function	Rule/Convention
Overstrikes	Overstriking characters to form special symbol operators may be performed in accordance with Table 2-1. (Either character is typed first, the carrier is backspaced once, and then the other character is typed.) Overstriking with an "X" or other character for character/text deletions is not allowed.
Visual Fidelity	It is not necessary to type complete line text or characters in order; entries are interpreted by the system only after the return function is completed. That is, the time sequence in which the operator presses the various keys doesn't matter; the system interprets the entry as it is viewed on the paper or display.
Local/Communicate Mode	The terminal must be in the remote, communicate mode to use the APL/700 system. The local mode may be used for off-line typing. (Switching between local and communicate modes may generate an extra transmitted character, often a "9".)

Other convenience keys, not always available on all terminals are as follows:

Linefeed	Acts like attention key, but does not provide a prompt for in-line editing.
Repeat	Provides repeated, automatic typing of any other character/symbol.

SIGN-ON PROCEDURE.

Perform the following procedure to activate the APL terminal and sign on to the APL/700 system:

NOTE

The following procedure assumes the use of an acoustic coupler-telephone communications interface. Minor variations to the procedure may be required for the various terminals.

1. Turn on terminal and acoustic coupler power.

2. Lift handset from telephone cradle, dial allocated computer telephone number, and listen for high-pitched tone.

3. When tone is received from computer, place handset in acoustic coupler cradle. (Position handset so that cord end is in coupler receiver marked CORD.)

4. Wait for sign-on response from computer; press return, attention, or break key if necessary. A typical response is as follows:

   ON-LINE TO APL/700; YOU ARE: N3163 (LSN:6).

   Where: N3163 and (LSN: 6) are variable telephone number and Logical Station Number, respectively.

   If necessary, repeat pressing ATTN key until response is received.

5. When above response is received, type following entry:

   \APL

6. Wait for system "prompt" (cursor types five-space indentation); then type following system command entry:

)ON Acct [Password]

   Where: Acct is user account identification.

   Password is optional entry for a previously locked account; omit entry if account is unlocked.

   Example:)ON RJS [BIGBE]

7. Press return (RTN) key and wait for system sign-on response, which has the following typical format:

   FRIDAY 74/02/01 11.51 AM [V25041 W120 T00 S006]

   Where: V25041 is the release version of APL/700 being used.

   W120 is the terminal width (numbers of characters per line) assumed for the user account.

   T00 is the terminal tab interval assumed for the account.

   S006 is the number of the port to which terminal is connected (used for communications purposes).

8. Wait for system prompt (five-space indention).

When a system prompt is received, the sign-on procedure is completed, the keyboard is unlocked, and the system is ready to process a user transaction entry. (The system provides the user with a block of its internal storage facilities called an active workspace.)

TRANSACTION PROCEDURE.

When the system sign-on process is completed, the user may perform a series of transactions (cycles consisting of user entry and system response) as follows:

1. Make certain that APL system has initiated cycle by typing a prompt (usually a five-space indention) and unlocking keyboard.

2. Using character/symbol keys and shift bar(s), type in desired entry. For example, if it is desired to set a workspace name, type the following:

)WSID Name

   Where: Name is desired workspace identifier.

   Example:)WSID WKSP01

3. When transaction entry is typed, press return (RETURN) key to signal system and initiate processing.

4. Wait for system to complete transaction (indicated by returning a typed response at left margin). (Response can be a statement, transaction result, or error message.)

NOTE

If an error message is received, perform the appropriate action/correction and reenter the desired transaction specification. (Refer to Section 8 for error-message descriptions and to the paragraphs describing editing and recovery procedures in this section.)

When the transaction is completed with a valid system response (printout or display), a five-space prompt is returned and the keyboard is unlocked to enable the next transaction entry. Repeat steps 2 through 4 for each subsequent transaction.

SIGN-OFF PROCEDURE.

When all user transactions are completed, or it is necessary to temporarily interrupt operations at the terminal, perform the following procedure to sign off from the system:

1. Make certain that a prompt (five-space indention) has been been typed and the keyboard is unlocked.

2. Type one of the following sign-off commands:

)OFF [Oldlock/Newlock]
   or
)COFF [Oldlock/Newlock]

   Where:)OFF is system sign-off command. (Work sesssion is terminated and active workspace is discarded.) If it is desired to save active workspace for continued use during the next session, enter ")COFF". [Oldlock/Newlock] is optional entry specifying account locks (passwords), if used.

   NOTE

   Oldlock is lock (password) presently used on account. Newlock is new lock assigned during sign-on process. If it is not desired to change lock, or there is no lock on account, omit entire Oldlock/Newlock entry. Leave Newlock entry blank to remove lock from account.

   Example:)COFF [BIGBE/PAOLI]

3. Press return key and wait for the following system sign-off statement (typical):

```
      )OFF
THURSDAY 74/01/17 02.42 PM
CONNECTED  00.55.48   TO DATE  02.06.20
CPU TIME   00.00.02   TO DATE  00.00.09
IN APL-MCS
```

   Where: This response reflects the type, time and date of sign-off; the four other numerical responses reflect time (hours, minutes, and seconds) spent on current session, plus total time to date for both connection and CPU usage. IN APL-MCS is version of message control system.

4. Remove telephone handset from acoustic coupler and return handset to telephone cradle.

5. Turn off terminal and coupler power as required.

RECOVERY OPERATIONS.

The APL/700 system provides automatic recovery from temporary work-session interruptions, accidental disconnections, or system malfunctions. When a user signs off from the system for a temporary work-session interruption (by using the ")COFF" command) the active workspace is returned when the next session is signed on.

When the active workspace is saved from a session and the account is signed on by using the previously described sign-on procedure, the system responds with the normal sign-on display with an additional statement as follows:

WS Name CONTINUED FROM Time Date

Where: Name is identifier of saved workspace.

Time and Date indicate time and date that previous work session was suspended.

Example:

```
      )ON RJS
WEDNESDAY 74/01/30 11.18 AM [V25041 W130 T05 S018]
CONTINUED FROM 74/01/30 11.02.33
```

It is possible that an accidental disconnection or system malfunction will occur during any work session. In either case, the system will automatically save the active workspace and provide a "CONTINUED" message when the account is again signed on.

If the execution of a program was interrupted, then the word "EXECUTION" will appear between the active workspace name and "CONTINUED". The program execution will continue until the line being executed is completed; then the function name and line number are printed, followed by an asterisk or star (*) symbol to indicate that the function is suspended. The system then types an input prompt (five spaces) and waits for a transaction entry.

If a function was being defined when a work session is interrupted, the word "DEFINITION" will appear between the workspace name and the word "CONTINUED" in the message. A function definition prompt is then returned to enable continuation of the function definition. An accidental interruption that occurs while an entry is being composed results in the loss of that entry attempt.

TRANSACTION EDITING.

There are a number of variations in performing transaction editing in the APL/700 system. The procedures required for editing depend on the mode of operation, on the state of the keyboard (locked or unlocked), on whether an "attention" entry is initial or non-initial, and on the type of editing required.

CORRECTING IN-PROCESS TYPING ERRORS. If a typing error is noticed before the transaction text entry is completed and the return (RETURN) key is pressed, the error is corrected as follows:

1. Using BACKSPACE and space keys, position cursor at the leftmost character that is in error.

2. Press attention (ATTN or BREAK) key and observe that system types inverted caret (∨) under character backspaced to in previous step and then feeds paper up one line. (This action effectively erases the errored character and every character to the right.) The INDEX key (when supplied on some terminals) can be used for this attention application; however the inverted caret is not viewed.

3. When system response is completed, resume typing correct entry and the remainder of transaction entry.

   Example: Suppose that the system command ")ON ACCT1 [KEYLOAD]" is typed, where "KEYLOAD" was intended to be "KEYLOCK". The resultant system response and correction is as follows:

```
)ON ACCT1 [KEYLOAD]
               ∨
              CK]
```

   The corrected entry ")ON ACCT1 [KEYLOCK]" is now entered by the carriage-return function.

TRANSACTION EDITING PROCEDURE. APL/700 has provisions for retrieving the most recently entered transaction and modifying it. This may be desired as part of a normal evolution of a computational expression, or in response to an error message or wrong result. The procedure for applying transaction editing is as follows:

1. Make certain that cursor is at five-space indention position, then press attention key.

2. Observe that previously entered transaction is displayed and that carrier returns to left margin.

3. Type edit symbols below corresponding displayed characters as follows (space cursor accordingly):

   "/" The foreslash causes deletion of character above it.

   "." Each period segments display into another phrase, starting with the character above the period. (The first phrase starts at the left of the line.)

4. Terminate edit line by pressing attention (not return).

5. Observe that first phrase is displayed (up to the first period of the edit line, or the entire remaining phrase if no period is used).

6. Alter or augment displayed phrase by normal typing.

7. Enter another attention at the right-most position to display the next phrase.

8. Repeat steps 6 and 7 until entry is complete. A transaction entry is completed by either entering a return (or an attention when no more phrases exist).

   Editing a line of a defined function involves the use of steps 3 through 8 of the above procedure.

Any transaction-entered character not recognized by APL/700 results in a "CHARACTER ERROR" report. An example of this and subsequent editing is as follows:

```
      'THIS ▯▯ A B▯D LINE.'
*** CHARACTER ERROR ***
      'THIS ▯▯ A B▯D LINE.'
           //.  ////
      'THIS IS  A LINE.'
                v
                FIXED LINE.'
THIS IS  A FIXED LINE.
```

All invalid characters are replaced by the squat quad (▯) symbol, and the cursor returns to the left margin. Editing can now be performed by starting at step 3 of the transaction editing procedure. If the attention key is pressed initially, a "twitch" prompt (three spaces and three backspaces) is returned.

If the return key is pressed during the above procedures, the entry is completed whether or not the immediate-edit procedure was completed. If the return key is pressed too soon, an appropriate error message is returned.

Entering any character other than a space, slash, or period below a line re-displayed for editing results in the following error message:

EDIT ERROR

Pressing the attention key permits the reinitiation of the transaction editing sequence.

The use of the attention key for in-process typing corrections does not conflict with the applications described above. That is, for in-process typing, the cursor is not at the rightmost entry position when the attention key is pressed.

EXAMPLES OF ATTENTION KEY APPLICATIONS. The following examples summarize the use of the attention key in transaction editing procedures. In the presentation, the left column indicates user entries, system responses, and the use of the attention, return, space, backspace, and edit functions. The right column contains annotations reflecting the various user and system actions.

NOTE

In the following examples:

Ⓡ indicates the return function.
@ indicates the attention function.
----- indicates user spacing.
← indicates user backspacing.

Example	Comments
'THIS IS A BAD LINE.' Ⓡ	User transaction entry.
*** *CHARACTER ERROR* *** *'THIS □□ A B□D LINE.'*	System response to invalid overstrikes not recognized as APL characters.
@-----	User-entered attention to extend cursor to left margin.
-----//. //// Ⓡ *'THIS IS A LINE.'* -----@ @ ← v *FIXED LINE.'* Ⓡ *THIS IS A FIXED LINE.*	User spaces to edit "IS", enters attention to accept remaining phrase (editing by "////"), backspaces to beginning of "LINE", enters another attention, retypes new phrase, and then enters a return to display the complete edited line.

Example	Comments
@	User enters attention for immediate editing of previous transaction entry.
'THIS IS A FIXED LINE.'	System responds with display.
----- . ®	User spaces to "F", types an edit period, and then enters a return.
'THIS IS A ® **** SYNTAX ERROR **** v *'THIS IS A*	System responds with first phrase, but user erroneously forgot remaining phrase and entered a return.
@ @ *REPAIR.'* ----- ® *'THIS IS A REPAIR.'*	Two attentions positions cursor at end of phrase, and remaining phrase is entered with a return to complete the editing procedure.

INPUT/OUTPUT COMMUNICATION FUNCTIONS.

APL/700 provides implicit input and output forms. Thus, user-typed entries are accepted as inputs; results not explicitly assigned to a variable are returned by the system as outputs to the user display.

The system also has specialized forms of input and output primitive functions related to communications between the system and the terminal. For example, there are times within a user defined procedure (function) where a value input from the user is required for evaluation to continue the operation/calculation. In this case, the evaluated-input function is embedded where needed in the defined function and results in an evaluated-input prompt to the user when that point is reached during execution.

Table 2-2 lists the format and definition of the various input/output communication functions. The quad (⎕) and quad-prime (⍞) symbols are used, along with value-producing expressions in these applications.

TABLE 2-2

INPUT/OUTPUT COMMUNICATION FUNCTIONS

NAME	FORM	DEFINITION
MIXED OUTPUT	E1;E2;E3;...	THE EXPRESSIONS EI ARE SEPARATELY EVALUATED RIGHT TO LEFT THEN THE RESULTS ARE OUTPUT. IF ALL THE EXPRESSIONS RETURN VECTORS AND SCALARS THEY ARE OUTPUT ON THE SAME LINE WITH NO SPACES BETWEEN THE END OF THE OUTPUT OF ONE EXPRESSION AND THE BEGINNING OF THE OUTPUT OF THE NEXT. MATRICES AND TENSORS ARE OUTPUT ON SEPARATE LINES.
EVALUATED INPUT	⍞	THE QUAD ('⍞') IS A PSEUDO-VARIABLE. WHEN IT IS USED WHERE A VALUE IS NEEDED THE PROMPT '⍞: ' FOLLOWED BY A LINEFEED IS SENT. THE USER MAY THEN ENTER ANY VALUE PRODUCING EXPRESSION. THE EXPRESSION WILL BE EVALUATED AND THE VALUE PRODUCED WILL BE USED IN PLACE OF THE QUAD. AN ESCAPE MAY BE MADE FROM EVALUATED INPUT BY ENTERING AN ABORT, '→'. THIS HAS THE SAME EFFECT AS IF THE ABORT WERE EXECUTED IN THE FUNCTION AT THAT POINT.
OUTPUT	⍞←E	WHEN THE QUAD IS USED TO THE LEFT OF THE REPLACEMENT THE VALUE BEING ASSIGNED TO IT IS OUTPUT. AS USUAL WITH ASSIGNMENT IT MAY BE IMBEDDED AS PART OF A LARGER EXPRESSION.
CHARACTER INPUT	⍠	QUAD-PRIME ('⍠') IS A VARIABLE SHARED WITH THE APL PROCESSOR, I.E. IT MAY BE LOCALIZED IN A DEFINED FUNCTION. WHEN QUAD-PRIME IS USED WHERE A VALUE IS REQUIRED THE USER DEFINED PROMPT IS OUTPUT AND THE KEYBOARD IS UNLOCKED. THE USER THEN ENTERS ANY STRING. THE QUAD-PRIME IS THEN REPLACED BY THE INPUT STRING PREFIXED WITH AS MANY BLANKS AS THERE ARE CHARACTERS IN THE PROMPT. AN ESCAPE MAY BE MADE FROM CHARACTER INPUT BY ENTERING ⍫ (O, BACKSPACE, U, BACKSPACE, T) AS THE FIRST INPUT ENTERED. THIS HAS THE SAME EFFECT AS ABORT FOR EVALUATED INPUT.
SET PROMPT	⍠←E	WHEN THE QUAD-PRIME APPEARS TO THE LEFT OF THE ASSIGNMENT THE VALUE ASSIGNED TO QUAD-PRIME IS THE PROMPT USED FOR ALL FUTURE QUAD-PRIME INPUT REQUESTS UNTIL THE PROMPT IS CHANGED. THE DEFAULT PROMPT IS THE EMPTY STRING(''). ONLY CHARACTER STRINGS MAY BE ASSIGNED TO QUAD-PRIME.
IMPLICIT OUTPUT	E	IF E IS A VALUE PRODUCING EXPRESSION AND THE LAST FUNCTION EXECUTED WAS NOT AN ASSIGNMENT PRIMITIVE THEN THE VALUE PRODUCED BY THE EXPRESSION WILL BE OUTPUT.

SECTION 3

THE APL/700 LANGUAGE

GENERAL.

The language of APL/700 is composed of an exceedingly simple syntax, together with very powerful primitive functions that apply to data structures. Data structures in APL/700 may range from simple elements to arrays. The primitive functions address these data elements in a parallel fashion, so that iteration is unnecessary. APL/700 has a large number of functions which may transform values or structures, or both. Furthermore, the user may define his own set of functions that are syntactically equivalent to the primitive functions of the language and that operate on the data structures in the same manner as these primitive operators. APL/700 has a distinctive symbol set instead of reserved words. The rules for composing expressions from these symbols are also simple, thus the notation for APL/700 is very concise and non-ambiguous. In the APL/700 language, the allocation of memory to a variable is done at the moment of assignment interpretation so that no previous declaration as to size, shape, or type is required.

LANGUAGE ELEMENTS.

APL/700 is used by entering statements or expressions which employ primitive functions, primitive operators, or defined functions intermixed with data objects. Primitive functions and operators are supplied by the APL/700 system, whereas defined functions are created and supplied by the user through the use of the function-definition process. User-defined functions can be niladic, monadic, or dyadic (no arguments, one argument, or two arguments, respectively) functions which do or do not return explicit results.

This section describes the primitive functions and operators and their applications. Section 4 provides a complete description of defined functions and associated definition, editing, and diagnostic operations. Elementary information concerning language data elements and order of expression execution is provided in the following paragraphs.

DATA ELEMENTS.

A data object or datum in APL/700 is defined in terms of its type, shape, rank, and value. The type of a datum is either numeric (Boolean, integer, or real), or character (any of the APL characters).

The shape of a datum is a vector with elements indicating the size of (number of elements along) each dimension.

The rank of a datum is a (one element) vector indicating the number of dimensions. Allowable ranks are 0, 1, 2, ... (through) 15. Names used to refer to rank structures are as follows:

Rank	Name
0	Scalar
1	Vector
2	Matrix
Any (1-15)	Array

Rank can be viewed in geometric terms: scalars as points, vectors as line segments, matrices as rectangles, rank-3 objects as rectangular solids, and so forth.

The general datum is an array characterized as follows:

a. Homogeneous (single type).

b. N-dimensional (rank N).

c. Rectangular. (All sub-arrays across a dimension have the same shape.)

d. Dense. (All elements have values, as contrasted with sparse in which some means is provided to indicate the coordinates of significant values.)

Table 3-1 shows examples of the various forms of data elements.

Note that the display of negative numeric data uses the "¯" symbol to the upper left of the number. This symbol is distinct from the subtract or negation symbol (-) in primitive function applications. For example:

```
¯2.5   0   3.2   ¯4.1
```

Also note that an exponential or "scientific" representation is provided to reflect the significant digits and the power of 10 multiple. For example:

```
¯387E3 is equivalent to ¯387000
1.2E¯3 is equivalent to 0.0012
```

Functions can be applied to arrays, subarrays, or individual elements of an array. Indexing is used to specify operation on subarrays or elements thereof. For example, in the following structure, element 13 has an index of [1;3].

```
11   12   13
21   22   23
```

Index [;2] with an "empty" prefix, is used to specify the entire dimension of column 2 in the above example. That is, all rows of column 2 in the above example. That is, all rows of column 2 are specified to yield a vector result of "12 22". Refer to the Indices reference sheet for additional information.

ORDER OF EXECUTION.

When primitive functions or user value-returning functions are combined with variables or constants to form a compound expression, it is important to make their order of execution clear. Because APL has so many functions, an attempt to establish precedence in determining the order of execution would result in unwarranted complexity. Instead, the following three general rules apply:

a. A function is evaluated only when the values of the quantities it requires for its evaluation (also referred to as arguments or parameters) are known.

b. The order of evaluating value producing functions in a compound expression is <u>right to left</u>.

c. Parenthesis are used in the conventional mathematical way to alter the order of execution.

Thus, a monadic function is evaluated when the value of its (right) argument is determined. A dyadic function is evaluated when both of its arguments (left and right) are determined. Because an argument can itself be an expression or a defined function, the process of determining the value of the expression or function is referred to as elaboration.

The order of argument elaboration for a dyadic function is generally unimportant; the order is usually right to left. An exception to this is where the right argument is a simple variable name. If elaboration of the left argument changes the value of the right argument, a surprising result occurs.

ORDER OF EXECUTION EXAMPLES. In the following example formats, "V" represents a value, "m" represents a monadic function, and "d" represents a dyadic function. Each elaboration of a value-producing function replaces the function and its argument(s) with a value. Each elaboration of an expression within parenthesis replaces it with a value.

```
VdVdV          (Expression)
Vd(VdV)        (Equivalent expression)
 2  1          (Order of elaboration)

VdmV           (Expression)
Vd(mV)         (Equivalent expression)
 2 1           (Order of elaboration)
```

```
(VdV)dV          (Expression)
  1  2          (Order of elaboration)

m(m(VdV)dmV)dVdmV      (Expression)
8 6  4  5 3 7 2 1      (Order of elaboration)
```

Note that is is not necessary to enclose right arguments within parenthesis; however extra parenthesis do not have any adverse effect. Some numerical representations to illustrate the order of execution are as follows:

Expression	Equivalent
$3\times5+2$ 21	$3\times(5+2)$ 21
$1-2-3$ 2	$1-(2-3)$ 2
$-_4^{(1-2)-3}$	-
$-_{10}^{5\times-2}$	$-_{10}^{5\times(-2)}$
$-_{10}^{5\times\bar{}2}$	$-_{10}^{5\times(-2)}$

Table 3-1

Examples of Data Object Forms

Numeric			Structure	Character		
Value (Example)	Rank	Shape		Value (Example)	Rank	Shape
1	0	(Empty)	SCALAR	A	0	(Empty)
¯2.5 0 3	1	3	VECTOR	*A∩⍸13$*	1	6
11 12 13 21 22 23	2	2 3	MATRIX	*TEXT* *TYPE*	2	2 4
111 112 121 122 131 132 211 212 221 222 231 232 311 312 321 322 331 332	3	3 3 2	ARRAY	*ABCD* *EFGH* *IJKL* *MNOP* *QRST* *UVWX*	4	2 3 1 4

PRIMITIVE FUNCTIONS AND OPERATORS.

APL/700 provides a set of statement or expression forming functions called standard or primitive because they are immediately available to the user for application. That is, once primitive functions are entered with specified arguments, a single result is obtained. These primitive functions and operators are grouped into the following application/function categories:

a. Assignment and selection functions.
b. Scalar monadic and dyadic functions.
c. Structure functions.
d. Mixed functions.
e. Set functions.
f. Compound operators.
g. Format functions.

This section contains a description of the form and application of each category.

ASSIGNMENT
FUNCTIONS

ASSIGNMENT PRIMITIVE FUNCTIONS

A VARIABLE IS A NAME WHOSE CURRENT MEANING IS NOT A LABEL, FUNCTION, OR GROUP. A NAME MUST BEGIN WITH A LETTER(A-Z, Å-Z, Δ, OR Δ) AND MAY BE FOLLOWED BY ANY NUMBER OF ADDITIONAL LETTERS, DIGITS(0-9), OR UNDERSCORES(_). A VARIABLE IS GIVEN A VALUE BY THE REPLACEMENT FUNCTION. THE REMAINING ASSIGNMENT PRIMITIVES REQUIRE THE VARIABLE HAVE A VALUE. A VARIABLE MAY BE ASSIGNED ANY APL DATA STRUCTURE OF ANY TYPE. INSERTION MAY ONLY INSERT VALUES OF THE SAME TYPE. WHEN THE REPLACEMENT PRIMITIVE IS USED ANY PREVIOUS VALUE THE VARIABLE HAD IS FORGOTTEN AND THE NEW VALUE IS ASSIGNED.

NAME	*FORM*	*DEFINITION*
REPLACEMENT	*V←E*	*V MUST BE A VALID VARIABLE NAME. THE VALUE E BECOMES THE VALUE ASSOCIATED WITH V. REPLACEMENT MAY BE IMBEDDED IN AN EXPRESSION. THE RESULT RETURNED BY REPLACEMENT IS THE VALUE ASSIGNED TO V, I.E. E.*
INSERTION	*V[L]←E*	*V MUST BE A VARIABLE THAT HAS BEEN PREVIOUSLY ASSIGNED A VALUE. L IS A SUBSCRIPT LIST FOR V. THE SPECIFIED ELEMENTS OF V ARE ASSIGNED THE CORRESPONDING VALUES FROM E. IF E IS A SCALAR THEN ALL SPECIFIED ELEMENTS ARE ASSIGNED THE SAME VALUE. THE RESULT OF INSERTION IS THE VALUES INSERTED, I.E. E.*
MODIFY	*V⍟←E*	*OPERATES IN THE SAME MANNER AS IF V←V⍟E HAD BEEN WRITTEN. MAY NOT BE USED WITH SYSTEM VARIABLES. ⍟ IS ANY SCALAR DYADIC FUNCTION. THE RESULT OF MODIFY IS THE VALUE ASSIGNED TO V, I.E. V⍟E.*
MODIFIED INSERTION	*V[L]⍟←E*	*OPERATES IN THE SAME MANNER AS IF V[L]←V[L]⍟E HAD BEEN WRITTEN. ⍟ IS ANY SCALAR DYADIC FUNCTION. CAN SAVE COMPUTER TIME IF THE SUBSCRIPT LIST L IS COMPLICATED. THE RESULT OF MODIFIED INSERTION IS THE VALUES INSERTED, I.E. V[L]⍟E*

SELECTION PRIMITIVE FUNCTION

FORM: V[E1;E2;...;EK]

WHERE:
V IS A VARIABLE, CONSTANT OR PARENTHESIZED VALUE PRODUCING EXPRESSION.
EI IS ANY VALUE PRODUCING APL EXPRESSION OR MAY BE EMPTY.
K MUST BE EQUAL TO THE RANK OF V OR THERE MUST BE 1 LESS THAN THE RANK OF V SEMICOLONS.

RESULT:
IF EACH EI IS A SCALAR THEN THE RESULT IS A SCALAR- THE SELECTED ELEMENT OF V. TO UNDERSTAND THE SELECTION PRIMITIVE CONSIDER A FUNCTION F WHICH WHEN GIVEN TWO SCALARS FORMS AN ORDERED PAIR AND GIVEN A SCALAR AND AN (N-1)-TUPLE FORMS AN ORDERED N-TUPLE. THEN THE N-TUPLE MAY BE USED TO SPECIFY AN ELEMENT OF AN N DIMENSIONAL ARRAY. APL DOES NOT REQUIRE EI BE A SCALAR. CONSIDER F AS THE FUNCTION IN AN OUTER PRODUCT, I.E. THE RESULT OF A∘.F B IS A STRUCTURE CONTAINING ALL POSSIBLE ORDERED PAIRS WHOSE FIRST ELEMENT IS FROM A AND WHOSE SECOND ELEMENT IS FROM B. THE OUTER PRODUCT IS FORMED IN THE NORMAL WAY. LET EACH SEMICOLON IN THE LIST BE REPLACED BY ∘.F AND FORM THE STRUCTURE OF ORDERED N-TUPLES. THEN SELECTION TAKES THE STRUCTURE OF ORDERED N-TUPLES AND REPLACES EACH N-TUPLE BY THE ELEMENT IT SELECTS FROM THE STRUCTURE IT IS INDEXING. EI MAY BE EMPTY IN WHICH CASE THE SUBSCRIPT IS TAKEN TO BE ⍳(⍴V)[I] WHERE V IS THE STRUCTURE BEING INDEXED AND I IS THE NUMBER OF THE DIMENSION FOR WHICH THE SUBSCRIPT WAS OMITTED.
SELECTION MAY APPEAR ON THE LEFT OF THE REPLACEMENT. ONLY THE SELECTED VALUES OF THE VARIABLE ARE REPLACED. THE STRUCTURE TO THE RIGHT OF THE REPLACEMENT ARROW MUST HAVE THE SAME SHAPE AS THE STRUCTURE OF ORDERED N-TUPLES OR MAY BE A SCALAR. THE SELECTED ELEMENTS ARE ASSIGNED THE VALUES. IF THE SAME ELEMENT IS SELECTED MORE THAN ONCE FOR REPLACEMENT THE RESULT IS UNDEFINED.

EXAMPLES:

LET

```
11 12 13 14
21 22 23 24 ↔ M          1 2 3 4 5 ↔ V
31 32 33 34
```

THEN

```
      22 ↔ M[2;2]              3 ↔ V[3]
   24 34 ↔ M[2 3;4]         1 3 5 ↔ V[1 3 5]
   11 13                 5 4 3 2 1 ↔ V[5 4 3 2 1]
   21 23 ↔ M[;1 3]       1 2 3 4 5 ↔ V[]
   31 33
31 32 33 34 ↔ M[3;]
      14 14              1 2 2 1 4 ↔ V[1 2 2 1 4]
   24 24 ↔ M[;4 4]
   34 34
```

DYADIC SCALAR PRIMITIVE FUNCTIONS $A \circledast B$	$\circledast$	*MONADIC SCALAR PRIMITIVE FUNCTIONS* $\circledast B$

DEFINITION OR EXAMPLE	*NAME*		*NAME*	*DEFINITION OR EXAMPLE*
1.5 ↔ ¯2+3.5 5.5 ↔ 2+3.5 ¯1.5 ↔ 2+¯3.5	*ADD*	+	*IDENTITY*	0+B ↔ +B 3.5 ↔ +3.5 ¯3.5 ↔ +¯3.5
¯1.5 ↔ 2-3.5 1.5 ↔ 3.5-2 5.5 ↔ 2-¯3.5	*SUBTRACT*	-	*NEGATE*	0-B ↔ -B ¯3.5 ↔ -3.5 3.5 ↔ -¯3.5
5 ↔ 4×1.25 ¯3 ↔ 6×¯.5 0 ↔ 0×¯.09	*MULTIPLY*	×	*SIGNUM*	*SIGN OF* B: 1 ↔ ×7.2 0 ↔ ×0 ¯1 ↔ ×¯3
1.76 ↔ 3.52÷2 ¯5 ↔ 10÷¯2 4 ↔ 12÷3	*DIVIDE*	÷	*RECIPROCAL*	1÷B ↔ ÷B .5 ↔ ÷2 ¯2 ↔ ÷¯.5
A *RAISED TO THE POWER* B: 9 ↔ 3*2 1024 ↔ 2*10 2 ↔ 4*.5 ¯3 ↔ ¯27*(÷3)	*POWER*	*	*EXPONENTIAL*	(2.71828...)*B 4 ↔ *1.386294361... 20.0855... ↔ *3
(⍟B)÷⍟A ↔ *LOGARITHM OF* B *FOR BASE* A ↔ A⍟B 1.87506... ↔ 10⍟75 3 ↔ 2⍟8	*LOGARITHM*	⍟	*NATURAL LOGARITHM*	(2.17828...)⍟B N ↔ *⍟N ↔ ⍟*N 1.386294361... ↔ ⍟4 .693147... ↔ ⍟.5
LARGER OF A *AND* B ↔ A⌈B 7 ↔ 3⌈7 6.01 ↔ 6.01⌈6.01 ¯3 ↔ ¯3⌈¯7	*MAXIMUM*	⌈	*CEILING*	*SMALLEST INTEGER NOT LESS THAN* B ↔ ⌈B 4 ↔ ⌈3.141 ¯3 ↔ ⌈¯3.141 101 ↔ ⌈101
SMALLER OF A *AND* B ↔ A⌊B 3 ↔ 3⌊7 6.01 ↔ 6.01⌊6.01 ¯7 ↔ ¯3⌊¯7	*MINIMUM*	⌊	*FLOOR*	*LARGEST INTEGER NOT GREATER THAN* B ↔ ⌊B 3 ↔ ⌊3.141 ¯4 ↔ ⌊¯3.141 101 ↔ ⌊101
3 ↔ 5|13 B-A×⌊B÷A ↔ A|B *FOR* A≠0 5 ↔ 0|5 B ↔ A|B *FOR* A=0 5 ↔ 14|5 .14 ↔ 1|3.14 4 ↔ 5|¯11 ¯1 ↔ ¯4|7	*RESIDUE*	|	*MAGNITUDE*	*ABSOLUTE VALUE OF* B ↔ |B 9.5 ↔ |9.5 9.5 ↔ |¯9.5 0 ↔ |0
6 ↔ 2!4 (!B)÷(!A)×!B-A *FOR* A≤B 0 ↔ 9!3 1 ↔ 5!5 0 ↔ A!B *FOR* A>B ¯10 ↔ 3!¯3 4.9346... ↔ 1.1!4.5	*COMBINATION*	!	*FACTORIAL*	B×!B-1 ↔ !B *FOR* B≥1, B *AN INTEGER*; 1 ↔ !0 *GAMMA*(B+1) ↔ !B *FOR NON-INTEGER* B 6 ↔ !3 39916800 ↔ !11 2.68344... ↔ !2.3 3.3283... ↔ !¯2.3
(1-B*2)*.5 ↔ 0○B (1-B*2)*.5 ↔ 0○B *ARCSIN* B ↔ ¯1○B *SIN* B ↔ 1○B *ARCCOS* B ↔ ¯2○B *COS* B ↔ 2○B *ARCTAN* B ↔ ¯3○B *TAN* B ↔ 3○B (¯1+B*2)*.5 ↔ ¯4○B (1+B*2)*.5 ↔ 4○B *ARCSINH* B ↔ ¯5○B *SINH* B ↔ 5○B *ARCCOSH* B ↔ ¯6○B *COSH* B ↔ 6○B *ARCTANH* B ↔ ¯7○B *TANH* B ↔ 7○B	*CIRCULAR*	○	*PI TIMES*	B×3.14159... ↔ ○B 6.283185... ↔ ○2
0⍟0 0⍟1 1⍟0 1⍟1 *BOOLEAN DOMAIN (0 OR 1)* 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0	*AND* *OR* *NAND* *NOR*	~ ∧ ∨ ⍲ ⍱	*NOT*	0 ↔ ~1 1 ↔ ~0 *BOOLEAN DOMAIN (0 OR 1)*
0⍟0 1⍟0 0⍟1 1⍟1 3⍟4 3⍟3 4⍟3 0 0 1 0 1 0 0 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1	*LESS* *NOT GREATER* *EQUAL* *NOT LESS* *GREATER* *NOT EQUAL*	< ≤ = ≥ > ≠		

EXTENSION OF DYADIC SCALAR PRIMITIVE FUNCTIONS AND MONADIC SCALAR PRIMITIVE FUNCTIONS TO ARRAYS

'⍟' IS ANY MONADIC SCALAR PRIMITIVE FUNCTION. '⍟' IS ANY DYADIC SCALAR PRIMITIVE FUNCTION.

⍟B

THE RESULT OF A MONADIC SCALAR PRIMITIVE FUNCTION APPLIED TO AN ARRAY IS AN ARRAY OF THE SAME SHAPE. ELEMENTS OF THE RESULT ARE THE RESULT OF APPLYING '⍟' TO THE CORRESPONDING ELEMENTS OF B.

A⍟B

IF A AND B ARE ARRAYS OF THE SAME SHAPE THEN AN ELEMENT OF THE RESULT IS THE RESULT OF APPLYING '⍟' TO THE CORRESPONDING ELEMENTS IN A AND B. THE RESULT HAS THE SAME SHAPE AS A. IF EITHER A OR B IS AN ARRAY CONTAINING ONLY ONE ELEMENT OR IS A SCALAR THEN AN ELEMENT OF THE RESULT IS THE RESULT OF APPLYING '⍟' BETWEEN THE ONE ELEMENT AND THE CORRESPONDING ELEMENT OF THE OTHER ARRAY. THE RESULT HAS THE SAME SHAPE AS THE ARRAY NOT CONTAINING THE ONE ELEMENT. IF BOTH ARRAYS CONTAIN ONE ELEMENT THEN THE RESULT IS A ONE ELEMENT ARRAY WITH RANK EQUAL TO THE LARGER OF THE RANKS OF A AND B.

A⍟[K]B

A AND B ARE ARRAYS WITH ONE OF RANK ONE GREATER THAN THE OTHER. THE SHAPES OF A AND B MUST BE THE SAME IF THE K TH DIMENSION OF THE ONE OF GREATER RANK IS ELIMINATED. THE RESULT HAS THE SAME SHAPE AS THE ARRAY OF LARGER RANK. ELEMENTS OF THE RESULT ARE FORMED BY APPLYING '⍟' BETWEEN THE CORRESPONDING ELEMENT OF THE ARRAY OF LARGER RANK AND THE ELEMENT OF THE ARRAY OF LESSER RANK WITH THE SAME INDICES EXCEPT FOR THE K TH WHICH IS NOT USED. THE SAME ELEMENT OF THE ARRAY OF LESSER RANK IS APPLIED TO ALL ELEMENTS OF A VECTOR ON THE K TH DIMENSION OF THE ARRAY OF LARGER RANK. THE '[K]' MAY BE ELIDED IF K IS EQUAL TO THE RANK OF THE LARGER ARRAY (IN ORIGIN 1).

STRUCTURE PRIMITIVE FUNCTIONS

THE RIGHT ARGUMENT OF ANY STRUCTURE PRIMITIVE FUNCTIONS MAY BE A CHARACTER STRUCTURE. SINCE CATENATE AND LAMINATE JOIN TWO STRUCTURES, IF THE RIGHT ARGUMENT IS A CHARACTER STRUCTURE THE LEFT ARGUMENT MUST ALSO BE A CHARACTER STRUCTURE. ALL OTHER STRUCTURE PRIMITIVE FUNCTIONS OPERATE IN THE SAME MANNER ON CHARACTER STRUCTURES AS ON NUMERIC STRUCTURES. FILL FOR TAKE AND EXPAND IS BLANKS IF THE RIGHT ARGUMENT IS A CHARACTER STRUCTURE.

THE FOLLOWING VARIABLES ARE USED IN THE EXAMPLES:

```
      1 2 3 4 5 ↔ V
11 12 13 14
21 22 23 24 ↔ M
31 32 33 34
```

```
111 112 113 114
121 122 123 124
131 132 133 134
                  ↔ T
211 212 213 214
221 222 223 224
231 232 233 234
```

NAME	FORM	DEFINITION	EXAMPLES
RESHAPE	A⍴B	THE STRUCTURE B IS MADE INTO THE SHAPE SPECIFIED BY A. IF B HAS LESS ELEMENTS THAN ARE NEEDED THE ELEMENTS OF B ARE REUSED UNTIL ENOUGH ELEMENTS ARE OBTAINED. IF B HAS MORE ELEMENTS THAN ARE NEEDED THE EXCESS ARE IGNORED. ,A ↔ ⍴A⍴B	5 5 5 ↔ 3⍴5 ,1 ↔ 1⍴V 2.5 ↔ (⍳0)⍴2.5 8.6 ¯3.1 1 2 3 4 ↔ 3 2⍴V 5 1
RAVEL	,B	THE STRUCTURE B IS RESHAPED INTO A VECTOR. ,B ↔ (×/⍴B)⍴B	11 12 13 14 21 22 23 24 31 32 33 34 ↔ ,M 1⍴8.6 ↔ ,8.6
CATENATE	A,B	THE STRUCTURES A AND B ARE JOINED TOGETHER TO FORM A NEW STRUCTURE. THE STRUCTURES ARE JOINED ALONG THE LAST DIMENSION. A SCALAR IS EXTENDED TO FORM A PLANE ACROSS THE DIMENSION IT IS BEING JOINED TO.	1 2 3 4 5 1 2 3 4 5 ↔ V,V 7 1 2 3 4 5 ↔ 7,V 7 11 12 13 14 8 21 22 23 24 ↔ 7 8 9,M 9 21 22 23 24 11 12 13 14 1 21 22 23 24 1 ↔ M,1 31 32 33 34 1
	A,[K]B	LIKE A,B BUT THE STRUCTURES ARE JOINED ON THE K TH DIMENSION.	11 12 13 14 21 22 23 24 ↔ M,[1]7 8 9 10 31 32 33 34 7 8 9 10
LAMINATE	A,[K]B	THE STRUCTURES A AND B ARE JOINED ALONG A NEW DIMENSION. K MUST BE NON-INTEGRAL AND BETWEEN THE NUMBERS OF THE DIMENSIONS BETWEEN WHICH THE NEW DIMENSION IS FORMED. A SCALAR IS EXTENDED TO THE SHAPE OF THE OTHER OBJECT.	1 100 2 200 ↔ 1 2 3,[1.5]100 200 300 3 300 1 2 3 4 5 8 8 8 8 8 ↔ V,[.476]8
REVERSE	⌽B	B VECTOR: THE ORDER OF THE ELEMENTS IN B IS REVERSED. B ARRAY: THE VECTORS ON THE LAST DIMENSION OF B ARE REVERSED. B SCALAR: NO ACTION OCCURS WHEN B IS A SCALAR.	5 4 3 2 1 ↔ ⌽V 14 13 12 11 24 23 22 21 ↔ ⌽M 34 33 32 31 1.5 ↔ ⌽1.5
	⌽[K]B	SAME AS ⌽B BUT VECTORS ON THE K TH DIMENSION ARE REVERSED.	31 32 33 34 21 22 23 24 ↔ ⌽[1]M 11 12 13 14
	⊖B	⊖B ↔ ⌽[1]B REVERSAL ALONG THE FIRST DIMENSION.	31 32 33 34 21 22 23 24 ↔ ⊖3 3⍴M 11 12 13 14
	⊖[K]B	⊖[K]B ↔ ⌽[1+(⍴⍴B)-K]B REVERSAL ALONG THE K TH FROM LAST DIMENSION.	14 13 12 11 24 23 22 21 ↔ ⊖[1]M 34 33 32 31

ROTATE	A⌽B	*B VECTOR: THE ELEMENTS OF THE VECTOR ARE ROTATED TO THE LEFT CYCLICALLY (⍴B)|A POSITIONS. B ARRAY: VECTORS ON THE LAST DIMENSION OF B ARE ROTATED BY THE AMOUNT SPECIFIED BY THE CORRESPONDING ELEMENT IN A. A MUST BE AN ARRAY OF RANK ONE LESS THAN THE RANK OF B AND SHAPE SAME AS B LESS THE LAST ELEMENT. A MAY BE A SCALAR IN WHICH CASE IT SPECIFIES THE ROTATION FOR ALL VECTORS. B SCALAR: NO OPERATION IS PERFORMED IF B IS A SCALAR.*	`3 4 5 1 2 ↔ 2⌽V` `4 5 1 2 3 ↔ ¯2⌽V` `14 11 12 13` `21 22 23 24 ↔ ¯1 0 1⌽M` `32 33 34 31` `12 13 14 11` `22 23 24 21 ↔ 5⌽M` `32 33 34 31` `5 ↔ ¯8⌽5`
	A⌽[K]B	*LIKE A⌽B BUT VECTORS ON THE K TH DIMENSION ARE ROTATED.*	`31 12 23 34` `11 22 33 14 ↔ ¯4 ¯3 ¯2 ¯1⌽[1]M` `21 32 13 24`
	A⊖B	A⊖B ↔ A⌽[1]B *ROTATION ON THE FIRST DIMENSION.*	`21 22 23 24` `31 32 33 34 ↔ 1⊖M` `11 12 13 14`
	A⊖[K]B	A⊖[K]B ↔ A⌽[1+(⍴⍴B)-K]B *ROTATION ON THE K TH FROM LAST DIMENSION.*	`12 13 14 11` `23 24 21 22 ↔ 1 2 ¯1⊖[1]M` `34 31 32 33`
TRANSPOSE	⍉B	*B SCALAR: THE RESULT IS B AS A 1 × 1 MATRIX. B VECTOR: THE RESULT IS B AS A COLUMN VECTOR(A ⍴B × 1 MATRIX).* *B ARRAY: THE RESULT IS B WITH THE DIMENSIONS REVERSED.* (⌽⍳⍴⍴B)⍉B ↔ ⍉B *FOR 2≤⍴⍴B*	`1 1⍴¯6.3 ↔ ⍉¯6.3` `¯1` `0 ↔ ⍉¯1 0 1` `1` `11 21 31` `12 22 32 ↔ ⍉M` `13 23 33` `14 24 34`
PERMUTE	A⍉B	*THE DIMENSIONS OF B ARE PERMUTED AS SPECIFIED BY A. THE I TH DIMENSION OF B IS THE A[I] DIMENSION OF THE RESULT. SEVERAL DIMENSIONS OF B MAY BE MAPPED INTO A SINGLE DIMENSION OF THE RESULT TO OBTAIN A DIAGONAL CROSS SECTION OF B. IF A IS THE SAME AS ⍳⍴⍴B THEN THE RESULT WILL BE B.*	`11 21 31` `12 22 32 ↔ 2 1⍉M ↔ ⍉M` `13 23 33` `14 24 34` `11 22 33 ↔ 1 1⍉M` `111 121 131 ↔ 1 2 1⍉T` `212 222 232` `1 2 3 4 5 ↔ 1⍉V`
COMPRESS	A/B	*B VECTOR: A MUST BE A LOGICAL VECTOR WHOSE LENGTH IS IS THE SAME AS THE LENGTH OF B. THE RESULT HAS LENGTH +/A. THE ELEMENTS OF THE RESULT ARE TAKEN FROM B EVERYWHERE A 1 APPEARS IN A. A MAY BE A SCALAR IN WHICH CASE THE RESULT IS B IF A IS 1 AND THE EMPTY VECTOR IF A IS 0. B ARRAY: VECTORS ON THE LAST DIMENSION OF B ARE COMPRESSED BY A.* *B SCALAR: B IS EXTENDED TO THE LENGTH OF THE VECTOR A AND THEN COMPRESSED BY A.*	`1 2 4 ↔ 1 1 0 1 0/V` `2 3 5 ↔ 0 1 1 0 1/V` `1 2 3 4 5 ↔ 1/V` `⍳0 ↔ 0/V` `12 13` `22 23 ↔ 0 1 1 0/M` `32 33` `5 5 5 5 ↔ 1 0 1 1 0 0 1/5` `¯4.5 ¯4.5 ↔ 0 1 0 0 1 0 0 0/¯4.5`
	A/[K]B	*LIKE A/B BUT VECTORS ON THE K TH DIMENSION ARE COMPRESSED.*	`11 12 13 14` `31 32 33 34 ↔ 1 0 1/[1]M`
	A⌿B	A⌿B ↔ A/[1]B *COMPRESS ON THE FIRST DIMENSION.*	`21 22 23 24` `31 32 33 34 ↔ 0 1 1⌿M`
	A⌿[K]B	A⌿[K]B ↔ A/[1+(⍴⍴B)-K]B *COMPRESS ON THE K TH FROM LAST DIMENSION.*	`11 12` `21 22 ↔ 1 1 0 0⌿[1]M` `31 32`

EXPAND	*A\B*	*B VECTOR: A MUST BE A LOGICAL VECTOR SUCH THAT +/A IS THE SAME AS THE LENGTH OF B. THE RESULT HAS THE SAME LENGTH AS A WHERE SUCCESSIVE ELEMENTS OF B ARE USED WHERE EACH 1 APPEARS IN A AND FILL IS INSERTED WHERE EACH 0 APPEARS.* *B ARRAY: VECTORS ON THE LAST DIMENSION OF A ARE EXPANDED BY A.* *B SCALAR: B IS EXTENDED TO LENGTH +/A AND THEN EXPANDED BY A.*	`0 1 2 0 3 4 5 0 ↔ 0 1 1 0 1 1 1 0\V` `11 0 12 13 0 14` `21 0 22 23 0 24 ↔ 1 0 1 1 0 1\M` `31 0 32 33 0 34` `0 0 0 0 7 7 7 0 ↔ 0 0 0 0 1 1 1 0\7`
	A\[K]B	*LIKE A\B BUT VECTORS ON THE K TH DIMENSION ARE EXPANDED.*	`11 12 13 14` `0 0 0 0` `21 22 23 24 ↔ 1 0 1 0 1\[1]M` `0 0 0 0` `31 32 33 34`
	A⍀B	*A⍀B ↔ A\[1]B* *EXPANSION ON THE FIRST DIMENSION.*	`0 0 0 0` `11 12 13 14` `21 22 23 24 ↔ 0 1 1 0 1⍀M` `0 0 0 0` `31 32 33 34`
	A⍀[K]B	*A⍀[K]B ↔ A\[1+(⍴⍴B)-K]B* *EXPANSION ON THE K TH FROM LAST DIMENSION.*	`0 11 0 12 0 13 14` `0 21 0 22 0 23 24 ↔ 0 1 0 1 0 1 1⍀[1]M` `0 31 0 32 0 33 34`
TAKE	*A↑B*	*B VECTOR: THE RESULT IS THE FIRST(LAST) |A ELEMENTS OF B IF A IS POSITIVE(NEGATIVE). IF |A IS GREATER THAN THE LENGTH OF B THEN FILL IS ADDED AT THE END(BEGINNING) OF B.* *B ARRAY: A MUST BE A VECTOR WHOSE LENGTH IS EQUAL TO THE RANK OF B. THE RESULT OF TAKE IS A CORNER OF THE ARRAY.* *B SCALAR: B WILL BE MADE INTO A ONE ELEMENT OBJECT WITH RANK THE SAME AS THE LENGTH OF A THEN THE TAKE IS DONE ON IT.*	`1 2 3 ↔ 3↑V` `3 4 5 ↔ ¯3↑V` `1 2 3 4 5 0 0 ↔ 7↑V` `0 0 1 2 3 4 5 ↔ ¯7↑V` `11 12 13` `21 22 23 ↔ 3 3↑M` `31 32 33` `0 0 0 0 0` `0 0 0 0 0` `11 12 13 14 0 ↔ ¯5 5↑M` `21 22 23 24 0` `31 32 33 34 0` `0 ¯3` `0 0 ↔ 2 ¯2↑¯3`
DROP	*A↓B*	*B VECTOR:THE RESULT IS B WITH THE FIRST(LAST) |A ELEMENTS OF B REMOVED IF A IS POSITIVE(NEGATIVE). IF |A IS GREATER OR EQUAL TO THE LENGTH OF B THE RESULT IS AN EMPTY VECTOR.* *B ARRAY: A MUST BE A VECTOR WHOSE LENGTH IS EQUAL TO THE RANK OF B. THE RESULT OF DROP IS A CORNER OF THE ARRAY.* *B SCALAR: B WILL BE MADE INTO A ONE ELEMENT OBJECT WITH RANK THE SAME AS THE LENGTH OF A THEN THE DROP IS DONE ON IT.*	`4 5 ↔ 3↓V` `1 2 ↔ ¯3↓V` `⍳0 ↔ 7↓V` `⍳0 ↔ ¯7↓V` `11 12` `21 22 ↔ 0 ¯2↓M` `31 32` `1 1 1⍴8 ↔ 0 0 0↓8` `0 1 0 1⍴¯1.75 ↔ 5 0 1 0↓¯1.75`

MIXED PRIMITIVE FUNCTIONS

NAME	FORM	DEFINITION	EXAMPLE
SHAPE	ρB	SHAPE PRODUCES A VECTOR WHICH IS THE SHAPE OF THE ARGUMENT. A ↔ ρAρB	,5 ↔ ρ¯2 ¯1 0 1 2 2 3 4 ↔ ρ2 3 4ρι24 ι0 ↔ ρ'A'
INDICES	ιB	B MUST BE A NON-NEGATIVE INTEGER SCALAR. THE RESULT IS A VECTOR OF LENGTH B OF THE FIRST B INTEGERS STARTING AT THE INDEX ORIGIN. ι0 ↔ THE EMPTY NUMERIC VECTOR. ιN ↔ (ιN-1),N IN ORIGIN 1.	1 2 3 4 5 ↔ ι5 ,1 ↔ ι1 0ρ0 ↔ ι0
INDEX	AιB	A MUST BE A VECTOR. THE RESULT IS A STRUCTURE WITH THE SAME SHAPE AS B. EACH ELEMENT OF THE RESULT IS THE INDEX IN A OF THE THE FIRST OCCURENCE OF THE CORRESPONDING ELEMENT IN B. IF THE ELEMENT DOES NOT OCCUR IN A THE RESULT IS 1+ρA(IN ORIGIN 1, ρA IN ORIGIN 0).	3 ↔ 4 7 10 22ι10 1 2 1 3 ↔ 'ABCABCDE'ι'ABAC' 4 2 4 1 ↔ ¯1 0 1ι10 0 ¯16 ¯1 4 4 4 ↔ 'ABC'ι1 2 3
QUOTE	▼B	THE RESULT IS A CHARACTER STRUCTURE WITH THE SAME SHAPE AS B EXCEPT THE LAST DIMENSION IS EXPANDED. THE RESULT IS A CHARACTER REPRESENTATION OF B.	'1 2 3' ↔ ▼1 2 3 'APL' ↔ ▼'APL'
FORMAT	A▼B	SEE FORMAT CHART.	
EXECUTE	⍎B	B MUST BE A CHARACTER STRING WHICH IS A VALID APL EXPRESSION. THE RESULT OF EXECUTE IS THE RESULT PRODUCED FROM THE EVALUATION OF THE EXPRESSION IF IT PRODUCES A RESULT. IF THE EXPRESSION DOES NOT PRODUCE A RESULT EXECUTE MUST BE THE LEFTMOST OPERATOR IN THE STATEMENT.	4 ↔ ⍎'2+2' 1 2 3 4 5 ↔ ⍎'ι5' 'APL' ↔ ⍎'''APL''' ¯2 ¯1 0 1 2 ↔ ⍎▼¯2 ¯1 0 1 2
REPRESENTATION	AτB	B SCALAR: IF A IS A VECTOR THE RESULT IS A VECTOR THE SAME LENGTH AS A. THE RESULT CONTAINS THE REPRESENTATION OF B IN THE NUMBER SYSTEM A. IF A IS AN ARRAY THEN THE RESULT IS THE REPRESENTATION OF B IN THE NUMBER SYSTEMS SPECIFIED BY VECTORS ON THE FIRST DIMENSION OF A. B ARRAY: THE RESULT WILL BE A STRUCTURE WITH SHAPE (ρA),ρB WHERE VECTORS ON THE FIRST DIMENSION OF THE RESULT ARE THE REPRESENTATION OF A SCALAR IN B IN THE NUMBER SYSTEM SPECIFIED BY A VECTOR ON THE FIRST DIMENSION OF A. FUNCTIONS IN A MANNER SIMILAR TO OUTER PRODUCT.	1 0 1 ↔ 2 2 2τ5 0 26 23 ↔ 24 60 60τ1583 1 0 0 3 ↔(3 2ρ4 5)τ17 1 2 1 1 0 0 0 1 1 0 ↔ 2 2 2τ4 7 3 0 0 1 1 0
BASE-VALUE	AιB	B VECTOR: IF A IS A VECTOR THEN THE RESULT IS A SCALAR WHICH IS THE BASE 10 VALUE OF THE VECTOR IN THE NUMBER SYSTEM SPECIFIED BY A. A MAY BE A SCALAR IN WHICH CASE IT IS EXTENDED TO THE LENGTH OF B. IF A IS AN ARRAY THE RESULT HAS SHAPE ¯1+ρA AND CONTAINS THE REPRESENTATION IN BASE 10 OF B IN THE NUMBER SYSTEM SPECIFIED BY A VECTOR ON THE LAST DIMENSION OF A. B ARRAY: THE RESULT IS AN ARRAY WITH SHAPE (¯1+ρA),1+ρB. THE RESULT IS SCALARS WHICH ARE THE BASE 10 REPRESENTATION OF VECTORS ON THE FIRST DIMENSION OF B IN THE NUMBER SYSTEMS SPECIFIED BY VECTORS ON THE LAST DIMENSION OF A. FUNCTIONS IN A MANNER SIMILAR TO INNER PRODUCT.	5 ↔ 2 2 2ι1 0 1 1583 ↔ 24 60 60ι0 26 23 15 ↔ 2ι1 1 1 1 22 30 38 ↔ (3 2ρ5 5 7 7 9 9)ι4 2 4 6 ↔ 2 2 2ι3 2ρ1 1 0 1 0 0

ARRAY-INVERSE	⌹*B*	*B MUST BE A MATRIX WITH NO MORE COLUMNS THAN ROWS. THE RESULT IS THE INVERSE OR GENERALIZED INVERSE OF THE MATRIX IF IT EXISTS.*	3.5 ¯1.5 0.5 ¯4 2 ¯1 ↔ ⌹3 3⍴(4⍴1),2 3 ¯2 ¯1 2 1.5 ¯0.5 0.5
ARRAY-QUOTIENT	*A*⌹*B*	*B MUST BE A MATRIX WITH NO MORE COLUMNS THAN ROWS. A IS EITHER A VECTOR WITH LENGTH EQUAL TO THE NUMBER OF ROWS IN B OR A MATRIX WITH THE SAME NUMBER OF ROWS AS B. THE RESULT IS THE SOLUTION TO THE SYSTEM OF LINEAR EQUATIONS WITH COEFFICIENT MATRIX B AND RIGHT HAND SIDE(S) A IF IT EXISTS. WHEN B HAS MORE ROWS THAN COLUMNS THE RESULT IS A LEAST SQUARES FIT FOR THE SYSTEM.*	¯1 1 ↔ 0 ¯1⌹2 2⍴1 1 2
GRADE-UP	⍋*B*	*B MUST BE A NUMERIC VECTOR. THE RESULT IS A SET OF INDICES THAT CAN BE USED TO ORDER B IN ASCENDING ORDER.*	2 5 4 1 3 ↔ ⍋8 0 9 5 0
GRADE-DOWN	⍒*B*	*B MUST BE A NUMERIC VECTOR. THE RESULT IS A SET OF INDICES THAT CAN BE USED TO ORDER B IN DESCENDING ORDER.*	3 1 4 2 5 ↔ ⍒8 0 9 5 0
ROLL	?*B*	*B MUST CONTAIN POSITIVE INTEGERS. THE RESULT IS A STRUCTURE LIKE B WITH EACH ELEMENT A RANDOM CHOICE FROM ⍳S WHERE S IS THE CORRESPONDING ELEMENT OF B.*	1 1 ↔ ?1 1
DEAL	*A*?*B*	*A AND B MUST BE NON-NEGATIVE INTEGERS WITH A NOT GREATER THAN B. THE RESULT IS A VECTOR OF LENGTH A THE ELEMENTS OF THE RESULT ARE A RANDOM SELECTION WITHOUT REPLACEMENT FROM ⍳B.*	,1 ↔ 1?1 ⍳0 ↔ 0?10

SET PRIMITIVE FUNCTIONS A • B

NAME	•	DEFINITION OR EXAMPLE
MEMBERSHIP	∈	A∈B ↔ ∨/A∘.=,B 1 0 1 1 ↔ 1 2 3 1∈1 3 5 1 ↔ 10∈10 20 30 0 ↔ 10∈40 50 60 0 1 1 ↔ 'BOY'∈'COUNTRY' 0 1 0 0 0 0 1 ↔ 'COUNTRY'∈'BOY'
UNION	∪	A∪B ↔ A VECTOR OF DISTINCT ELEMENTS FROM A OR B 1 2 3 4 5 6 ↔ 1 2 3∪4 5 6 1 2 3 4 5 ↔ 1 2 3 4∪2 3 4 5 'BOYCUNTR' ↔ 'BOY'∪'COUNTRY' 1 2 3 ↔ 1 1 2∪1 3 3 'COUNTRYB' ↔ 'COUNTRY'∪'BOY' 'MANGET' ↔ 'MANAGEMENT'∪''
INTERSECTION	∩	A∩B ↔ A VECTOR OF DISTINCT ELEMENTS IN BOTH A AND B 2 3 ↔ 1 2 3∩2 3 4 'HAR' ↔ 'HARRY'∩'MARTHA' 'APL' ↔ 'APPLIED'∩'PLAN'
EXCLUSION	~	A~B ↔ A VECTOR OF DISTINCT ELEMENTS IN A BUT NOT IN B ,1 ↔ 1 2 3~2 3 4 ,'B' ↔ 'BOY'~'COUNTRY' 'TNC' ↔ 'PLATONIC'~'PAOLI' 'MISP' ↔ 'MISSISSIPPI'~''
SUBSET	⊂	A⊂B ↔ ∧/,A∈B 1 ↔ 1 2⊂4 3 2 1 0 ↔ 1 2⊂2 3 4 5 1 ↔ 'PAOLI'⊂'PLATONIC'
SUPERSET	⊃	A⊃B ↔ ∧/,B∈A 1 ↔ 2 2 3 3 4 4⊃2 3 4 0 ↔ 2 2 4 4 6 6⊃2 3 4 0 ↔ 'PAOLI'⊃'PLATONIC'

COMPOUND OPERATORS

NAME	FORM	DEFINITION	EXAMPLES
REDUCTION	⍟/B	B VECTOR: SCALAR RESULT IS FORMED BY EXECUTING THE APL EXPRESSION FORMED BY PLACING ⍟ BETWEEN THE ELEMENTS OF THE VECTOR. IF B IS AN EMPTY VECTOR THE RESULT IS THE IDENTITY ELEMENT FOR ⍟ IF IT EXISTS. B ARRAY: RESULT IS FORMED BY REDUCING VECTORS ON THE LAST DIMENSION OF THE ARRAY. THE RESULT HAS RANK 1 LESS THAN THE RANK OF THE ARGUMENT. THE SHAPE OF THE RESULT IS THE SAME AS THE SHAPE OF THE ARGUMENT LESS THE LAST DIMENSION. B SCALAR: THE RESULT IS THE SCALAR B. B MUST BE IN THE DOMAIN OF ⍟.	6 ↔ +/1 2 3 1.4 ↔ -/2.3 5.6 4.7 1 ↔ ×/⍳0 ¯4.31...E68 ↔ ⌈/⍳0 1.5 4.8 7.875 ↔ ⍟/3 3⍴⍳9 5 ↔ *⍟5
	⍟/[K]B	LIKE ⍟ BUT VECTORS ON THE K TH DIMENSION ARE REDUCED.	1.75 3.2 4.5 ↔ ⍟/[1]3 3⍴⍳9
	⍟⌿B	⍟⌿B ↔ ⍟/[1]B REDUCTION ON THE FIRST DIMENSION.	1.75 3.2 4.5 ↔ ⍟⌿3 3⍴⍳9 6 ↔ +⌿1 2 3
	⍟⌿[K]B	⍟⌿[K]B ↔ ⍟/[1+(⍴⍴B)-K]B REDUCTION ON K TH FROM LAST DIMENSION.	1.5 4.8 7.875 ↔ ⍟⌿[1]3 3⍴⍳9
SCAN	⍟\B	B VECTOR: RESULT IS A VECTOR OF THE SAME LENGTH WHOSE I TH ELEMENT IS ⍟/I↑B. B ARRAY: RESULT IS FORMED BY REPLACING VECTORS ON THE LAST DIMENSION OF B BY THE ⍟ SCAN OF THE VECTOR IN B. B SCALAR: THE RESULT IS THE SCALAR B. B MUST BE IN THE DOMAIN OF ⍟.	1 3 6 ↔ +\1 2 3 2.3 ¯3.3 1.4 ↔ -\2.3 5.6 4.7 1 0.5 1.5 4 0.8 4.8 ↔ ⍟\3 3⍴⍳9 7 0.875 7.875 1 ↔ ∧\1
	⍟\[K]B	LIKE ⍟\ BUT VECTORS ON THE K TH DIMENSION ARE SCANNED.	1 2 3 0.25 0.4 0.5 ↔ ⍟\[1]3 3⍴⍳9 1.75 3.2 4.5
	⍟⍀B	⍟B ↔ ⍟\[1]B SCAN ON THE FIRST DIMENSION.	1 2 3 0.25 0.4 0.5 ↔ ⍟⍀3 3⍴⍳9 1.75 3.2 4.5
	⍟⍀[K]B	⍟⍀[K]B ↔⍟\[1+(⍴⍴B)-K]B SCAN ON THE K TH FROM LAST DIMENSION.	1 0.5 1.5 4 0.8 4.8 ↔ ⍟⍀[1]3 3⍴⍳9 7 0.875 7.875
INNER PRODUCT	A⍟.⊗B	ELEMENTS OF THE RESULT ARE FORMED BY TAKING CONFORMING VECTORS ON THE LAST DIMENSION OF A AND THE FIRST DIMENSION OF B APPLYING ⊗ BETWEEN THEM AND REDUCING THE RESULT BY ⍟. M1+.×M2 IS THE LINEAR ALGEBRA MATRIX PRODUCT FOR MATRICES M1 AND M2.	32 ↔ 1 2 3+.×4 5 6 1 ↔ 1 0 1∨.∧1 1 0 5 6 7 8 ↔ (2 3⍴⍳6)-.⌈3 4⍴⍳12 8 8 8 8
OUTER PRODUCT	A⍟.⊗B	THE RESULT IS THE OPERATOR ⊗ APPLIED BETWEEN ALL PAIRS OF ELEMENTS SELECTED FROM A AND B. THE RESULT HAS SHAPE (⍴A),⍴B.	4 5 8 10 ↔ 1 2 3⍟.×4 5 12 15 0 1 1 1 ↔ 0 1⍟.∨0 1

⍟ AND ⊗ ARE ANY DYADIC SCALAR PRIMITIVE FUNCTIONS.
K IS A DIMENSION NUMBER OF B.

IDENTITIES FOR SCALAR DYADIC FUNCTIONS

THE RESULT OF REDUCTION OF AN EMPTY VECTOR IS THE IDENTITY(IF IT EXISTS) FOR THE FUNCTION. REDUCTION OF A DIMENSION OF LENGTH ZERO WILL PRODUCE AN ARRAY CONTAINING THE IDENTITY FOR THE FUNCTION. INNER PRODUCT AND BASE VALUE ARE DEFINED IN TERMS OF REDUCTION AND CAN CREATE ARRAYS CONTAINING THE IDENTITY. THE IDENTITIES FOR THE SCALAR DYADIC FUNCTIONS ARE LISTED BELOW ALONG WITH WHETHER THE IDENTITY IS A LEFT IDENTITY, RIGHT IDENTITY, OR TWO-SIDED IDENTITY.

●	*IDENTITY*	*LEFT, RIGHT, OR BOTH*
+	0	*BOTH*
-	0	*RIGHT*
×	1	*BOTH*
÷	1	*RIGHT*
*	1	*RIGHT*
⍟	*NONE*	
⌈	¯4.31359146674E68	*BOTH*
⌊	4.31359146674E68	*BOTH*
|	0	*LEFT*
!	1	*LEFT*
○	*NONE*	
∧	1	*BOTH*
∨	0	*BOTH*
⍲	*NONE*	
⍱	*NONE*	
<	0	*LEFT*
≤	1	*LEFT*
=	1	*BOTH*
≥	1	*RIGHT*
>	0	*RIGHT*
≠	0	*BOTH*

FORMAT
FUNCTIONS

FORMAT FUNCTIONS.

Formatted character data structures can be produced using the format primitive functions. The monadic form provides a default format. The dyadic forms permit explicit specification of the desired format. The discussion common to all forms or comparing forms is contained here; detailed differences are described on subsequent pages.

Forms:

⍕ E	Default format
V ⍕ N	Numeric explicit format
C ⍕ E	Character explicit format
C ⍕(L)	Character explicit formatted list

Where: E is a data object of numeric or character type
N is a numeric data object
V is a numeric vector defining the edit format
C is a character string defining the edit format
L is a list of APL data objects, separated by semicolons; each object is of type character or numeric.

Actions/Results:

The result is a character data object that represents the data objects(s) of the right argument, formatted as specified.

Shape: The default and numeric explicit forms preserve all dimensions except the last dimension and expand it if E is numeric. E may be a vector, matrix or general array.

Each character explicit form accepts as arguments only scalar, vector, or array data objects and results in a character matrix having at least one row, and generally the maximum number of rows of any matrix in the list.

Conditions/Options:

All formatting is constrained by the)WIDTH setting.

For explicit formatting, the numeric form is more efficient where appropriate than the character form. The character form however, has many more capabilities.

FORMAT SYNTAX DIAGRAMS.

The syntax of the format functions is displayed using syntax diagrams. This method shows the syntax clearly and concisely, including by connectivity the allowable constructs: defaults, alternatives, and iterations. It is rigorous without being cumbersome.

The rules for interpreting these diagrams are simple:

syntactic units are set off by spaces and separated by lines;

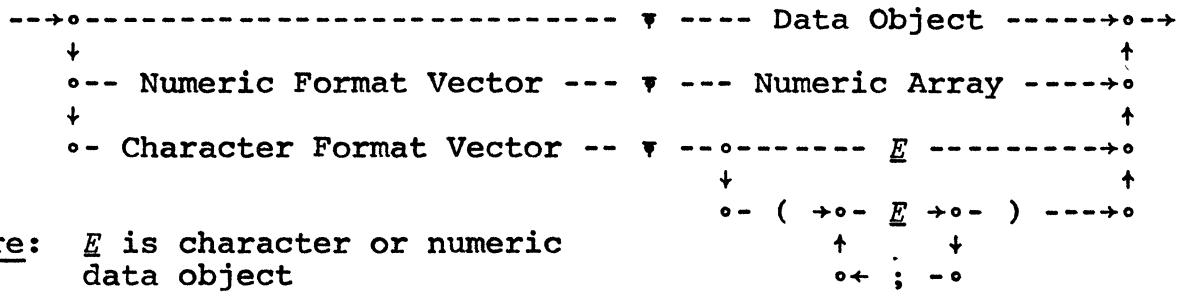
any path traced along a forward direction of the arrows will produce a syntactically valid command;

iteration is achieved by a leftward path;

limited iteration is shown by a "bridge" covering a number indicating the maximum number of traversals of the bridge;

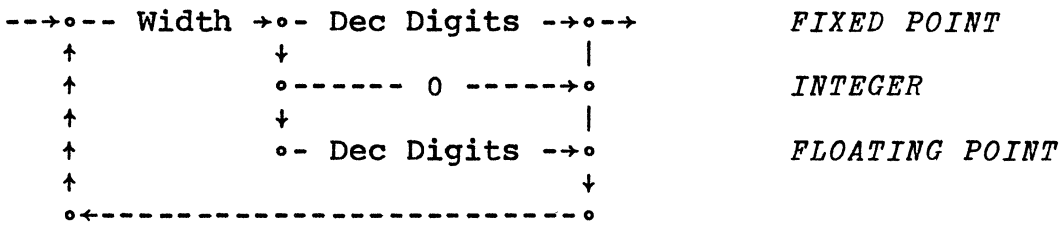
a vertical bar is a bidirectional path.

Format:



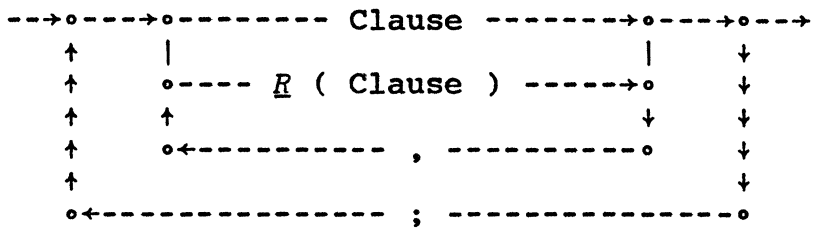
Where: *E* is character or numeric data object

Numeric Format Vector:



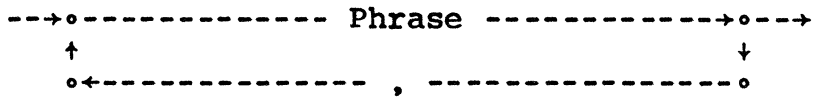
FORMAT
FUNCTIONS
(CONT.)

Character Format Vector:

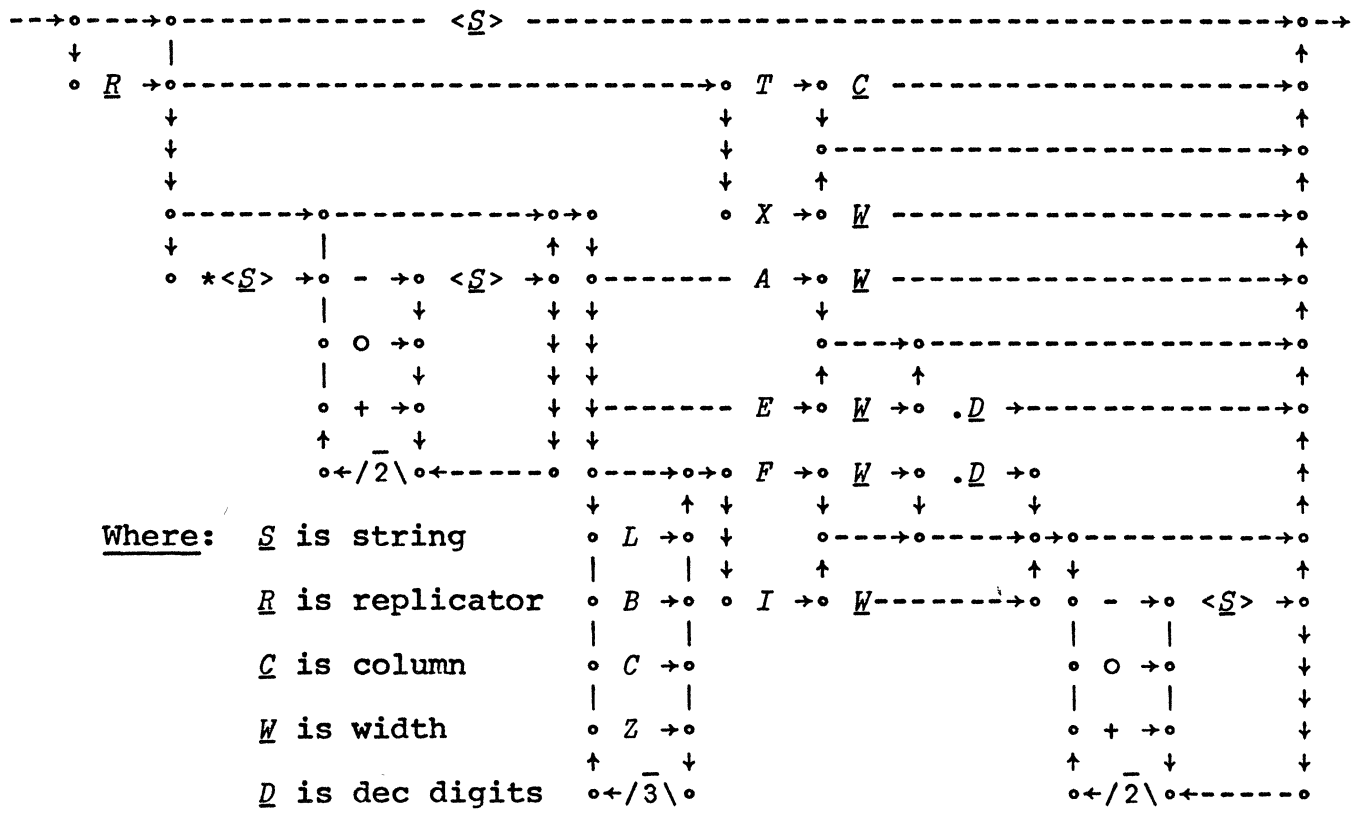


Where: *R* is clause replicator

Clause:



Phrase:



Where: *S* is string
R is replicator
C is column
W is width
D is dec digits

DEFAULT FORMATTING.

Default formatting is a monadic mixed primitive function that results in a character data structure.

Form:

▼E

Where: E is a data object

Actions/Results:

The result is a character data structure.

If E is of character type, the result is identically E.

If E is of numeric type, the result is formed as follows:

Every element of E is rounded according to the current print precision to get the specified number of significant fractional digits (integers are not truncated) and trailing fractional zeros are ignored and then converted to characters.

If E is scalar, one blank is prefixed.

If E is vector, the result is also a vector. This result is the ravel of an array formed containing the character representation of each element. Sufficient columns are provided that at least one blank precedes each non-blank, and all decimal points are aligned.

If E is an array, the result is also an array except that the last dimension is expanded in the same manner as if the array were raveled.

Conditions/Options:

The length of the last dimension of the result is an integer multiple of the length of the last dimension of E.

For some element(s) there will be only one preceding blank. Others may have more than one blank.

Print precision limits the printed numbers.

Exponential notation is used for all output if any element has either an integer part too big to be exactly expressed, or only a fractional part and the exponential notation would be shorter by 3 or more characters than the numeric notation.

FORMAT
FUNCTIONS
(CONT.)

Default Formatting Examples:

```
      ▼0 11 2222 ¯3333 0.4444
      0          11        2222       ¯3333           0.4444

      ▼⍉0 11 2222 ¯3333 0.4444
      0
     11
   2222
  ¯3333
      0.4444

      ▼'APL'
APL
```

NUMERIC EXPLICIT FORMATTING.

The numeric explicit format mixed primitive function provides efficient formatting of a numeric data object of any rank and produces a character data object.

Form:

V ▼ N

Where: V is numeric format vector
N is numeric data object to be formatted

Actions/Results:

The numeric data object N is represented as a character data object. The shape of the result is the same as N, except that the last dimension is determined by the format V.

The format V must be a vector of length 2xM where M is a positive integer. Successive pairs of elements from U specify how successive columns of N are to be formatted.

If W is the first and D is the second member of a pair, an element on the last dimension of N is formatted as follows:

D > 0	*F*W.D
D = 0	*I*W
D < 0	*E*W.D

If M is less than the last dimension of N, then the format V is cyclically reused.

Conditions/Options:

A field width inadequate to allow representation of the number is filled with '*'.

FORMAT
FUNCTIONS
(CONT.)

Numeric Vector Formatting Examples:

```
      10 3▼0 123 0.0125 ¯1234.5678
      0.000   123.000     0.012 ¯1234.568

      10 3▼⍉0 123 0.0125 ¯1234.5678
      0.000
    123.000
      0.012
  ¯1234.568

      5 0 5 0 8 4 12 ¯3▼0 123 0.0125 ¯12345.678
     0  123  0.0125    ¯1.235E4

      5 0 8 3▼2 2 2⍴111 112 121 122 211 212 221 222
   111 112.000
   121 122.000

   211 212.000
   221 222.000

      5 0 5 2▼ 3 5.12 8 27.3456 ¯5←1    3 5.12    827.35   ¯5

      8 ¯1▼⍉¯53.8 ¯0.0000345 0 12345678 4.0E¯15 ¯2.5E¯25
   ¯5.4E1
  ¯3.4E¯5
    0
    1.2E7
  4.0E¯15
 ¯2.5E¯25
```

CHARACTER EXPLICIT FORMATTING.

The character explicit format dyadic mixed primitive function provides the most general formatting capability.

Forms:

C ⍠ E	Character explicit format
C ⍠ (L)	Character explicit formatted list

Where: C is a character string specifying the format.
E is a data object of rank at most 2.
L is a list of data objects separated by semicolons.

There is no type or shape conformability requirement between list members.

Actions/Results:

The result is a character data matrix, that represents the right arguments according to the format specification.

The method is to create a character structure of appropriate shape filled with blanks. Then, into appropriate positions non-blank characters are inserted according to the format string as applied to corresponding portions of the right argument.

The number of rows in the result is the maximum of the number of rows in matrices that comprise the right argument. If only scalars or vectors appear in the right argument, then a matrix with one row results.

Conditions/Options:

The maximum rank of any data object in L is 2.

Separate formatting for each row of output takes place.

The fields in the result for any matrices with less rows than the maximum, are displayed with blanks.

Character Format Syntax Chart:

The format character string f has many options. It should conform to the following syntax. The leftmost entry is the syntactic unit being defined in terms of one of the alternatives, if any, to the right of 'is'. Upper or lower case letters in this type font represent

syntactic units further defined. Letters or characters in the APL font represent themselves. 'text' represents any APL text. Blanks are ignored except within 'text'. Character representations of integers are used for r, M, W and D.

```
f  is  s  or  s;s; . . . ;s                    format
s  is  g  or  g,g, . . . ,g  or empty          segment
g  is  c  or  r(c)                             group
r  is  optional clause replicator,             replicator
       default is infinite
c  is  p  or  p,p, . . . ,p                    clause
p  is  one of:                                 phrase

    M J A W          character object formatting
    M J E W.D        floating point numeric formatting
  M L Q F W.D R      fixed  point numeric formatting
  M L Q I W R        integer numeric formatting
      M X W          skip W characters forward
      M T N          tab to N characters from start of format;
                     (may be used to back up for replacement)
    <text>           literal text for each row;
                     may not contain the '>' character

M is  optional phrase replicator               phrase replicator
      default is 1
W  is optional total field width               field width
      default is 1
D  is optional number of places to right       decimal places
      of decimal point, default is 0
L  is B  or  C  or  B  C  or  empty            left decorator
B  is *<text>                                  background for field
R  is C  or  empty                             right decorator
C  is S<text>  or  S<text> S<text>             conditional text
      or  S<text> S<text> S<text>
S  is one or more of:                          sign selector

  -    insert 'text' in field if negative      sign selectors
  o    insert 'text' in field if zero
  +    insert 'text' in field if positive

J  is L  or empty, default is right            justifier left
      justify in field
Q  is zero or more of:                         qualifier

  L    left justify in field
  B    skip if zero
  C    insert commas
  Z    leading zero insert

N  is columns to right of start of format      next column
```

Character Format Semantics:

The prior syntax chart provides named syntactic elements for semantic description only. The terminal forms (shown in APL font) are the same as in the syntax diagram.

Any data object that provides a part of the right argument is generally the result returned from evaluation of an APL expression. In the following discussion, the general case is that a data object is treated as a matrix. A vector or scalar is treated as a matrix with only one row.

The form using a parenthesized list containing data objects separated by semicolons imposes no conformability or type restriction on adjacent data list members. The formatted result will have as many rows as there are in the data object having the most rows. The corresponding fields for objects with less fields will be blank. Each semicolon represents a synchronizing point with a semicolon in the corresponding format.

Each format segment applies in order to the corresponding data list member. The format segments are cyclically reused if necessary, until the entire data list has been formatted. If the format segment is empty, default formatting is used to format that data object.

Each format group applies in order to the corresponding columns of any one data list member. The format group is cyclically reused if necessary, until all columns of the data list member are formatted.

Within the format group an integer clause replicator can be used to limit replication. Without the replicator the clause is assumed to replicate cyclically as often as necessary.

A format clause is a series of phrases separated by commas.

Each phrase specifies the field width, and the content for that field resulting from either conversion of a data object or a literal text.

A The character object formatting phrase permits expansion between the columns of the object if W is greater than 1. It also permits choice of left or right justification.

E The floating point numeric formatting phrase provides results in scientific notation: mantissa E exponent, e.g., ¯3.2*E*¯2 or 9.73*E*21.

Default columns for non-negative signs are elided. This format can be explicitly justified left, or right by default.

F The fixed point numeric formatting phrase provides fixed, aligned format with a specified number of decimal places. This phrase permits qualifiers and left or right decorators.

I The integer numeric formatting phrase provides integer results with qualifiers and left or right decorators.

FORMAT
FUNCTIONS
(CONT.)

Any numeric formatting phrase for which the field width is too small gives '*' for the entire field in the row in which the data element was out of range.

X The skip formatting phrase provides rightward skip over the indicated number of columns. The replicator is not needed. Instead, using the default replicator of 1, the width can be the product of replicator times width. The columns are skipped, not blanked, to allow any prior content to remain.

T The tab formatting phrase allows absolute repositioning to any result column starting from the leftmost as column 0. Any subsequent formatting phrase will overwrite any prior contents.

A <text> phrase allows the indicated text to be unconditionally included in every row of the result. The text cannot contain the ' ' character.

R The integer phrase replicator specifies the number of uses of the phrase before moving to the next phrase in the clause.

W The total field width for character or numeric phrase formatting should include sufficient columns for the entire anticipated result range of values including signs and decorations.

D The decimal places for fixed point and floating point numeric formatting permit specified precision result display. Rounding occurs as part of formatting.

Left and/or Right Decorators can be applied to fixed point or integer formatting.
- o + The sign selectors alter the result depending on the signum of each individual data element. These prefixes to explicit text can be applied separately, or in pairs. In any one formatting phrase, only one occurrence of each sign selector should occur on each side. The same sign selector may appear in the left and right decorators. A '-' selector removes the sign from any negative element.

*<text> A field background can be specified. It is initally inserted in the field, then partially replaced.

L The default justification of phrases that do not require the specified width is to the right. Unless background is specified, excess columns to the left are blanked. Left justification can be explicitly specified instead, in which case excess columns to the right are blanked.

L B C Z Qualifiers alter the field content for integer and fixed point formatting. They include left justification; skipping (the numeric result and decimal point) if the element value is zero; insertion of leading zeros to fill the field; and insertion of commas to set off positive powers of 1000 for large numeric results.

Character Vector Formatting Examples:

Numeric data objects

```
      □←NV←¯1230 4.55 0 ¯0.465 60.525
¯1230  4.55  0  ¯0.465  60.525

      □←NM←¯0.05 25∘.×410 1 0.025
¯2.050E1    ¯5.000E¯2   ¯1.250E¯3
 1.025E4   2.500E1    6.250E¯1
```

Floating Point

```
      'E10.2'▼NV
   ¯1.23E3     4.55E0     0   ¯4.65E¯1     6.05E1

      'E9.4,E6.0,E10.2'▼NM
 1.0250E4   3.E1    6.25E¯1

      'E6.1'▼¯0.12 0.12
************

      'E7.1'▼¯0.12 0.12
¯1.2E¯1 1.2E¯1
```

Fixed Point

```
      'F10.2'▼⍉NV
  ¯1230.00
 4.55
 0.00
¯0.47
60.53

      'F10.2'▼NV
  ¯1230.00 4.55 0.00¯0.4760.53

      'F7.2,F6.1,F8.4'▼NM
 ¯20.50    0.0  ¯0.0013
*******   25.0   0.6250
```

Integer

```
      'I6'▼NV
 ¯1230 5   0    0    61

      'I5,I2'▼NM
  ¯21 0     0
1025025     1
```

FORMAT
FUNCTIONS
(CONT.)

Phrase Replicator

```
      '2I3,2I5,3I2'▼ 1 2 3 4 5 6 7 8 9
  1  2    3    4 5 6 7  8  9
```

Justify Left

```
      'LI5'▼◊¯1 2 34 567
¯1
2
34
567
```

Background

```
      '*<∘>I5'▼¯1 0 2
∘∘∘¯1∘∘∘∘0∘∘∘∘2

      '*</ \>I5'▼1 23 456
/ \/1/ \23/ 456
```

Sign Selectors

```
      '+<P>0<Z>-<N>I5'▼¯1 0 2
   M1  Z0  P2

      '+0< >-<(>I5+0< >-<)>'▼◊¯1 0 2
  (1)
   0
   2

      '+<        >0<    >LI12'▼◊¯1 32 0 541 ¯35
         ¯1
32
    0
541
        ¯35
```

Blank Zero Field

```
      'BI5'▼¯1 0 5
   ¯1    5

      'BLI5'▼◊¯1 0 5
¯1

5
```

Comma Insert

```
      'CI10'▼1234567
 1,234,567
      'CF12.4'▼1234.5678
  1,234.5678
```

Zero Insert Left

```
      'ZI3'▼⍉1 23 456
001
023
456
```

Combined

```
      '*<∘>ZBI5'▼⍉1 23 ¯456 0 987
00001
00023
¯0456
∘∘∘∘∘
00987

      'ZBCI7'▼1 0 2345 ¯1
000,001    002,345¯00,001

      'BCI5'▼⍉¯1 0 234 5678
   ¯1

  234
5,678
```

Character

```
      'A2'▼2 4⍴'GOODWORK'
 G O O D
 W O R K

      'A1,A2,A3,A4'▼'OPEN'
O P  E   N

      'LA2'▼'LEFT'
L E F T
```

FORMAT
FUNCTIONS
(CONT.)

Tab and Skip

```
      'I15,T0,I5,X20,I5'⍕25 50 75
   50         25              75

      'I15,T0,I5,I25'⍕25 50 75
   50                         75
```

Text

```
      '<|>,I5,<|∘>'⍕⍉1 10 ¯25

|     1|∘
|    10|∘
|   ¯25|∘
```

Combined

```
      'I5;X4,2A1'⍕(5 6;'AB')
    5    6    AB

      'I5;X4,A2'⍕(100;⍉'AB')
  100     A
  100     B

      'I5;X4,A1'⍕(,100;⍉'AB')
  100    A
         B

      'I5;A5;F5.1'⍕(⍉1 10 100;⍉'FINE';2 3⍴1.1×1 2 3 4 5 6)
    1    F  1.1  2.2  3.3
   10    I  4.4  5.5  6.6
  100    N
         E

      'LI5,2(LI3,F7.2,X4),I3'⍕3 5 15.72 17 23.15 ¯3
3    5    15.72    17   23.15     ¯3
```

SECTION 4

FUNCTION DEFINITION, EDITING, AND EXECUTION

FUNCTION DEFINITION AND EDITING.

Table 4-1 summarizes the function definition, opening, and closing commands and the full set of function editing commands. The definition of a function is started by typing the del (∇) symbol, which initiates entry into the function definition mode from the calculator mode. (Entry of the del symbol to initiate the definition mode may only be done while in the calculator mode.) The del symbol is then followed by the function specification header, which consists of two parts: (1) the function template, and (2) the local names (variables) list.

The function template may be entered in six different forms, depending on whether the function does or does not return a result and whether the function is niladic (no arguments), monadic (one argument), or dyadic (two arguments). The six forms of templates are as follows:

	NILADIC	*MONADIC*	*DYADIC*
RETURNS RESULT	*R←F*	*R←F B*	*R←A F B*
NO RESULT	*F*	*F B*	*A F B*

Where:

R is a dummy variable whose value is used as the value of the function when the function execution is completed.

B is a dummy variable that is initialized to the value of the right argument when execution of the function is started.

A is a dummy variable that is initialized to the value of the left argument when execution of the function is initiated.

F is the function name.

R, B, and A must be distinct names.

Table 4-1

Function Definition and Editing

COMMAND	ACTION SYMBOL	FORM	ACTION	NEXT PROMPT
DEFINE	∇	∇*H*	DEFINE NEW FUNCTION, HEAD OF WHICH IS *H*; INITIATE EDITING THEREON.	[1]
OPEN	∇	∇*F*	INITIATE EDITING OF PREVIOUSLY DEFINED FUNCTION, *F*.	[*Z*]
OPEN (LOCKED)	⍢	⍢*F*		
CLOSE	∇	∇	TERMINATE FUNCTION EDITING. (MAY FOLLOW ANY COMMAND EXCEPT EDIT)	
CLOSE (LOCKED)	⍢	⍢		
REPLACE		[*A*]*T*	TEXT OF LINE *A* IS REPLACED BY *T*. (IF *A* = *Z*, SAME AS APPEND-AFTER)	[*Z*]
APPEND-BEFORE	↑	[↑]*T*	TEXT OF NEW FIRST LINE IS *T*.	
APPEND-AFTER	↓	[↓]*T*	TEXT OF NEW LAST LINE IS *T*.	
INSERT-BEFORE	↑	[↑*A*]*T*	TEXT OF NEW LINE, TO BE INSERTED BEFORE LINE *A*, IS *T*.	[↑*A*+1]
INSERT-AFTER	↓	[↓*A*]*T*	TEXT OF NEW LINE, TO BE INSERTED AFTER LINE *A*, IS *T*.	[↓*A*+1]
FULL-EDIT	∈	[∈*A*]	INITIATE EDIT OF LINE *A*. RULES SAME AS FOR TRANSACTION EDIT.	[*Z*]
PREFIX-EDIT	⍺	[⍺*A*]	SIMILAR TO FULL-EDIT EXCEPT SINGLE INSERTION BEFORE TEXT OF LINE *A* IS ASSUMED.	
DIRECT-PREFIX	⍺	[⍺*A*]*T*	THE TEXT *T* IS INSERTED BEFORE THE TEXT OF LINE *A*.	
SUFFIX-EDIT	⍵	[⍵*A*]	SIMILAR TO FULL-EDIT EXCEPT SINGLE INSERTION AFTER TEXT OF LINE *A* IS ASSUMED.	
DIRECT-SUFFIX	⍵	[⍵*A*]*T*	THE TEXT *T* IS INSERTED AFTER THE TEXT OF LINE *A*	
IMMEDIATE-EDIT	⍳	[⍳*A*]	UPON TERMINATION OF DEFINITION MODE, TEXT OF *A* BECOMES THE 'MOST RECENT APL EXPRESSION' AVAILABLE FOR EDIT.	

Table 4-1 (cont)

Function Definition and Editing

DELETE	~		*DELETE THE D SELECTED LINES WITHIN RANGE R (DELETE ON LINE ZERO DELETES LOCAL NAMES LIST).*	[Z-D]
DISPLAY-LINES	⍞	*UNQUALIFIED: SELECTS EVERY LINE WITHIN THE INCLUSIVE RANGE R.*	*FORM OF DISPLAY:* *HEAD -* ∇H *BODY -* [N] T *TAIL -* ∇ H = HEAD OF FUNCTION N = LINE NUMBER T = LINE TEXT	[Z]
DISPLAY-TEXT	⍕		*FORM OF DISPLAY:* *HEAD -* ∇H *BODY -* T *TAIL -* ∇ H = HEAD OF FUNCTION T = LINE TEXT	
DISPLAY-CONTROLS	⎕	*FORM* | *RANGE* [o] | 0-Y [oA] | A [Ao] | A-Y [AoB] | A-B	*FORM OF DISPLAY:* *HEAD -* ∇C P *BODY -* [N] C P *TAIL -* ∇ C = CONTROLS SET τ - TRACE ⌈ - STOP ∩ - STATISTICS P = PROCESSOR UNITS	
DISPLAY-NUMBERS	?	*QUALIFIED: SELECTS ONLY LINES WITHIN THE INCLUSIVE RANGE R WHICH CONTAIN NAME X.*	*FORM OF DISPLAY: VECTOR OF NUMBERS*	
SET-TRACE *CLEAR-TRACE*	τ ⊥	*FORM* | *RANGE* [(oX)] | 0-Y [(oX)A] | A [A(oX)] | A-Y [A(oX)B] | A-B	*FORM OF DISPLAY (DURING EXECUTION):* F[N]K(S)V *LINE [0] TRACES FUNCTION RETURN.* F = FUNCTION NAME N = LINE NUMBER K = TYPE OF VALUE N - NUMERICAL B - BOOLEAN C - CHARACTER S = SHAPE OF VALUE V = VALUE	[Z]
SET-STOP *CLEAR-STOP*	⌈ ⌊	o = *ACTION SYMBOL* A,B = *ADDRESS(LINE NUMBER, LABEL OR LABEL* ∓ *INTEGER.*	*FORM OF DISPLAY (DURING EXECUTION):* F[N]* F = FUNCTION NAME N = LINE NUMBER *LINE [0] STOPS BEFORE RETURN.*	
SET-MONITOR *CLEAR-MONITOR*	∩ ∪		*INITIATE COLLECTION OF STATISTICS.* *LINE [0] COUNTS THE NUMBER OF TIMES THE FUNCTION IS EXECUTED.*	

The local names (variables) list is separated from the function template by a semicolon (;) and consists of a distinct set of names themselves separated by semicolons. System variables may be included in the local names list. Local names have meanings only while the function is active; that is, while the function is being executed. As soon as the execution of a function is terminated, the meaning of all local names disappears. The dummy variables for the result and argument(s) must be distinct from all other local names. The function specification is considered line 0 of the function, although not identified as such in the display.

After the user types the function specification and enters it by use of the carriage-return function, the system responds by typing the identifier for line 1 [1] and indents five spaces to prompt the user to enter the first line of the function. A user entry at this point is interpreted as the definition of line 1 of the function/program. After the first line is successfully entered by a carriage-return function, the next prompt is [2]. This operation is repeated for each subsequent line to be entered.

Definition editing commands (table 4-1) may be entered any time the function definition is open (immediately after a prompt). The function is terminated (closed) after all lines are completed by typing another del (∇), either at the end of the last line entered or after the last prompt is returned. If the del is entered at the end of an entry, it is not considered part of that entry.

Entries to a function may contain a label. The syntax of a function entry is:

FE is LS	Function Entry
LS is ID:LS or S	Labeled Statement

Where: ID is a string starting with a letter, underscored letter, del, underscored del, and is followed by any number of additional letters or underscored letters, numbers, dels, underscored dels, or underscores (_).

S is an expression or comment.

DEFINED FUNCTION EXECUTION.

The execution of a function begins when the function is first called, either from the calculator mode or from another function. From the instant execution of a function begins until the execution of the function is completed, the function is active. An active function may be suspended or pendant. A pendant function is one which is awaiting completion of a function it called. A suspended function in one which is currently executing or one whose execution was stopped for some reason other than a call to another function.

Execution of a function may be halted abnormally by an error, a stop on a line, or when the user presses the attention key.

The command ')SI is used to display the state indicator. The state indicator is a list of all suspended and pendant functions. The display of the state indicator has the following form: '$\underline{F}[\underline{N}]$'; where $\underline{F}$ is the function name and N is the line on which the function is suspended or pendant. If the function is suspended a star (*) will appear in the state indicator display; for example, 'F[6] *' indicates $\underline{F}$ is suspended on line 6.

While execution of a function is suspended, the user may examine and modify the contents of local variables, modify the suspended function, continue execution, or abort execution of the suspended function. A pendant function cannot be modified. No local names may be added or deleted from a suspended function. To continue execution of the suspended function, the user types '→$\underline{N}$', where $\underline{N}$ is the line in the function on which execution is to be continued. $\underline{N}$ is usually the line on which execution was stopped, but need not be. To abort execution of a function the user types '→'. The system responds by typing the name and line number of the next suspended function in the state indicator. If no more functions are suspended, no response is given to the abort.

It is good practice to eliminate all suspensions soon after they occur, as suspended and pendant functions take up space in the workspace. The user should avoid executing a function from the beginning after execution is stopped. If it is impossible to continue execution of a function after it is stopped the abort should be entered before execution of the function is begun again.

EXECUTION CONTROL.

The normal flow of execution of the lines in a defined function is sequential; that is the execution starts with line 1 and then proceeds to line 2 when line 1 is completed, and so forth. This sequence may be altered by two user-implemented functions: branching and aborting.

One way of altering the normal flow of defined function execution is by use of the branch primitive operator in the following form:

→$\underline{V}$

Where: $\underline{V}$ is any scalar or vector-valued expression.

There are many situations in which it is desirable to branch to some other line of a defined function other than the next one in sequence. For example, after a particular sequence of lines has been executed, it might be desirable to repeat them with a different set of values. It may also be desirable to repeat just one line, or all lines starting with a particular line.

The branch primitive shown above produces no result other than altering the sequence of execution. The branch operation may appear only at a point where no value is needed; that is, as the left-most function on a line, or the left-most function in a string to be evaluated where no result is required. The next line to be executed is the one specified by the scalar value or the first element of the vector. If there is no such line in the function, then execution of the

function is terminated and control is reestablished at the point where the function was called. If the specified vector has no first element, then control passes to the next line. Because function editing may cause line numbers to change, labels may be used to prevent the necessity of changing branches every time a line is added or deleted. A label is any valid identifier which prefixes a line and is separated from the statement by a colon(':'). A label is considered to be a local constant having the value of the line number it appears on. The name of a label must be distinct from all other local names.

A branch entered while the system is in the calculator mode causes execution of the last suspended function to be continued at the line specified.

The formats for the various methods of branching are as follows:

```
→L                          GO TO L
→(C)/L                      IF C THEN GO TO L
→(1+×E)⌽L1,L2,L3            BRANCH ON SIGN OF E(- L1, 0 L2, + L3)
→(L1,L2,L3,...)[N]          GO TO LN
→N+⍳26                      BRANCH N LINES FORWARD(BACKWARD IF N<0)
→0                          RETURN
```

Where:

L, L1, L2,... ARE LABELS OR LINE NUMBERS.
C IS A BOOLEAN EXPRESSION OR CONDITION.
E IS ANY SCALAR VALUED EXPRESSION.
N IS A SCALAR INTEGER VARIABLE OR CONSTANT.

It is also possible to abort execution of all functions up to and including the calculator mode entry that initiated the first function call leading to the suspension. The abort primitive entry from is the right arrow (→).

The abort function produces no result, but aborts the execution of all functions up to and including the calculator mode entry which initiated the first function call leading to the current suspension. Abort may be executed in calculator mode or as part of a function.

SECTION 5

SYSTEM COMMANDS

GENERAL.

APL/700 has a set of special instructions called system commands. These commands concern such practical matters as signing onto and off of the system, saving workspaces, setting default control values, copying workspaces, functions, or variables, and controlling terminal functions. These operations are only initiated in calculator mode; they can not appear as part of a user defined function. A system command is executed immediately after being entered (if possible).

SYSTEM COMMAND FORMAT.

The conventions used to describe the system commands are chosen to allow ready recognition of the fixed and variable, necessary and optional parts.

Representation	Meaning
)	System command prefix
[ ] ( ) /	Separators -- matching pairs for [ ] and ( )
COMMAND	Upper-case is required literal word
Variable	Initial Capitals
Variable	Underscore is optional variable name,
n	Number
n	optional number

The optional phrases or numbers change the meaning of the basic command. A command without an optional part is often an inquiry. The optional part provides a value or name for more detailed specification.

SYSTEM COMMAND CATEGORIES.

The system commands are grouped under the following categories:

Session Controls
Terminal Controls
Clear Workspace Controls
Library Controls
Name Displays
Run State
Group Commands

```
)BLOT
)WIDTH
)TABS
```

)BLOT obscure area

Blot provides multiple overprinting of an 18 character area, then backspaces to the prompt position to obscure subsequent display of a sensitive entry such as the Password on the account.

TERMINAL CONTROLS.

An account can be used from any terminal. The line width and tab setting must be specified.

)WIDTH n maximum characters in display line.

The number of characters n is in the inclusive range 30 to 130. If n is not specified, the result is the current width.

```
     )WIDTH 65
WAS 120
     )WIDTH
IS 65
```

)TABS n physical tab interval

The integer n is the number of characters between physical tab settings. This single interval should match the tabs as actually set on the terminal. If n is not 0, then output with "white space" will automatically use tabs to minimize the time to reach a position on the display.

```
     )TABS 5
WAS 0
     )TABS
IS 5
```

)*CLEAR*
)*SYMS*
)*ORIGIN*

CLEAR WORKSPACE CONTROLS.

Workspace controls provide the default SYMS, ORIGIN, DIGITS, SEED, and FUZZ for a clear workspace that is suited to the normal desires of the account user.

)CLEAR n clears workspace

The clear command without n destroys the prior active workspace and replaces it with a clear workspace having no names in it and the default attributes hereafter described. If n is specified, it refers to the number of symbols reserved for the symbol table. This number may be in the inclusive range 16 to 1024.

```
     )CLEAR
CLEAR WS
     )CLEAR 300
WAS 256
```

The response indicates the number of symbols in the prior active workspace. It does not change the default number, which is controlled by)SYMS.

The following commands return current values or allow the establishing of new default values for clear workspace attributes. The examples illustrate the installation provided default values and samples of changes to them.

)SYMS n default symbol table size

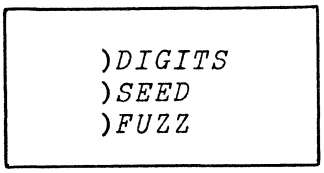
The default symbol table size for a clear workspace is set to n, in the inclusive range 16 through 1024. Since the space consumed in the workspace is 6 bytes per entry in the symbol table it should be controlled in space limited applications.

```
     )SYMS
IS 256
     )SYMS 400
WAS 256
```

)ORIGIN n default ordinal index origin

Origin affects primitive functions that use ordinal numbering. The default index origin can be overridden by the index origin system function ⎕*IO*.

```
     )ORIGIN
IS 1
     )ORIGIN 0
WAS 1
```



)DIGITS n default print precision

The default number of significant digits displayed in either fractional or exponential form is established in a clear workspace by the value of n. This must be an integer from 1 through 12 inclusive. The default digits can be overridden by the system function ⎕*PP*, print precision.

```
      )DIGITS
IS 10
      )DIGITS 4
WAS 10
```

)SEED n default random number seed

The quasi random number generator used in the roll and deal primitive function is pre-set to the default value of Seed. This permits repeated execution to receive the same supplied random values of an algorithm if desired. The value of n is a non-negative integer: 0 through 549755813887 (the largest integer). The seed is the starting value for the random link. The random link changes with each use of roll or deal and can be changed by the system function ⎕*RL* , random link.

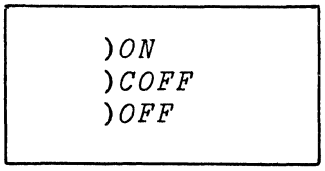
```
      )SEED
IS 0
      )SEED 377
WAS 0
```

)FUZZ n default comparison tolerance

The comparison tolerance by which two approximate representations of a number are considered equal is established in a clear workspace by)FUZZ n. The allowable range for n is 0 through 1. The default fuzz may be overridden by the system function ⎕*CT* comparison tolerance.

See that description for details.

```
      )FUZZ
IS 1E¯10
      )FUZZ 1E¯1
WAS 1E¯10
```



SESSION CONTROLS.

Session controls are used to initiate and terminate a work session.

)ON Accountname [Password]	signs on account
)COFF [Oldpassword/Newpassword]	signs off to continue
)OFF [Oldpassword/Newpassword]	signs off

)ON logs the account on the APL/700 system and initiates work. If any continuation workspace exists, it is reactivated at the point at which it was interrupted.

)COFF logs the account off, retaining the active workspace for reactivation at next)ON for that account.

)OFF logs the account off and discards the active workspace, so at next)ON for that account, the user will have a clear workspace.

Both)OFF and)COFF return date and time, then the amount of CPU (processor) time and elapsed time used. These amounts are given both for the session and cumulative for the installation accounting period. Units are hours, minutes, and seconds.

The Accountname is assigned by the installation. It is considered to be public knowledge.

The optional Password permits a user to protect his own account from unauthorized use. The Password can be initially set by the installation, or by the user at any sign off. Once set, a Password must be used for any successful sign-on. Until removed either Oldpassword or Newpassword may be empty. The forms for adjusting the password at signoff are:

[/Newpassword]	establishes password
[Oldpassword/Newpassword]	changes password
[Oldpassword/]	removes password

An Accountname may have 1 to 6 characters; a Password 1 to 12. These characters are alphanumeric (excluding the APL underscore alphabet).

*)ON BUR*103
)COFF[/SESAME]
*)ON BUR*103*[SESAME]*
*)OFF BUR*103 *[SESAME/AL*23814*]*

)*LOAD*
)*COPY*
)*PCOPY*
)*SAVE*

)LOAD Workspacename load copy of workspace

A copy of the specified workspace becomes the active workspace. The Wsid of the loaded workspace (not the Account or Password) becomes the name of the active workspace.

)*LOAD TEXTEDIT*
)*LOAD MYWORK[MYLOCK]*
)*LOAD (LIB) NEWS*

)COPY Workspacename Namelist replace copy

Copy into the present active workspace from the library workspace identified by Workspacename. If Namelist is present, copy only the names in it that are present in that workspace. If Namelist is absent, copy all functions, variables and groups in the workspace. A copied object will replace a prior object of the same name in the active workspace.

) *COPY TEXTEDIT*
) *COPY NEW [VERSION] FORECAST SCHEDULE*

)PCOPY workspacename Namelist protect copy

Same as)COPY except that any name in Namelist already existing in the active workspace will not be copied.

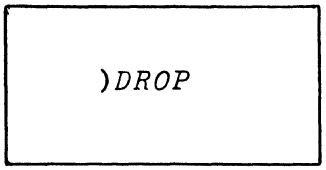
)*PCOPY (LIB)NEWS SCHEDULE INDEX*

)COPY and)PCOPY are more expensive commands than)LOAD.

)SAVE Wsid [Password] save workspace

A copy of the active workspace can be saved in the account library of the user. If Wsid is present, that name is the one used for subsequent library reference; if absent, the prior active workspace identifier is used. This will replace a former like-named workspace. If the Password is present, subsequent)LOAD or)COPY of that library workspace must supply the password.

)*SAVE NEW [VERSION]*



)DROP Wsid [Password] drop account library workspace

A workspace in the account library can be destroyed by using)DROP. The password is required if the workspace is locked. A workspace can not be dropped from any other account.

)DROP NEW[VERSION]

```
)FILES
)LIB
```

LIBRARY CONTROLS.

The library of an account includes named files and workspaces. Commands to interrogate the names and to totally or selectively access workspaces are provided. File access is done through primitive file system operators.

)FILES display account file names

The names of files owned by the account are returned. Only the public part of the name is displayed; any password on a file is omitted.

```
      )FILES
DATAFILE
DOCUMENT
```

)LIB display account library names

The identifiers of workspaces in the account library (but not their passwords) are displayed.

```
      )LIB
NEW
TEXTEDIT
```

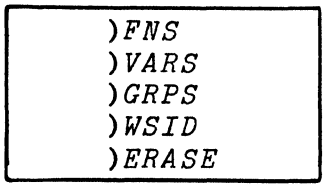
The form for referencing workspaces in the following)LOAD,)COPY, and)PCOPY commands is:

Workspacename is (Account) Wsid [Password]

The Wsid is the identifier by which the workspace is known. It must start with a letter followed by 0 to 11 letters or digits.

The Account portion is the owning account; the account library in which the workspace resides. It may be elided if it is in the user's own account.

The Password is used only if the workspace is locked. The password is also a name starting with a letter and followed by 0 to 11 letters or digits.



NAME DISPLAYS.

The following system commands display classes of names currently in the symbol table:

)FNS Name	display function names
)VARS Name	display variable names
)GRPS Name	display group names

If name is absent, the entire class is displayed in alphabetical order. If Name is present, only the members of the class starting with (or after) Name are displayed. The display result can not be used as an APL data object. The system function ⎕*NL*, name list, should be used for that purpose.

)WSID Name	workspace name

The workspace name provides a reference for the workspace when saved in the account library. The clear workspace is unnamed.

```
      )WSID
IS UNNAMED WS
      )WSID NEW
WAS UNNAMED WS
      )WSID
IS NEW
```

ERASE NAMES.

)ERASE Nameset	erase namelist

Names of functions, variables and primary names of group names in Nameset are erased from the workspace. The names in Nameset are entered, separated by spaces. Function names can not be erased while the state indicator is non-empty. Notice is given for non-existent or non-erasable members of its nameset. See discussion in Group commands following:

```
      )ERASE W X Y Z
NOT Y
NOT Z
```

```
)SI
)RESET
```

RUN STATE.

The run state is the record of user defined functions in process, suspended, or pending completion of other called functions.

)SI state indicator

The result is the stack indicating the run state of suspended and pending functions. The first line (if non-empty), is the most recently suspended function. Below are pending functions (awaiting completion of functions above) and earlier suspended function.

Each line gives function name, bracketed line number at which execution is pending or suspended, and, an asterisk for suspended functions only.

```
      )SI
RUN[1]*
MAIN[5]
RUN[4]*
TEST[6]*
```

A function can appear more than once in the state indicator. In line 5, MAIN called RUN. MAIN is pending completion of RUN. More than one suspended function can appear. A function can reappear (independent restarts, or recursive calls are permitted).

Usually the state indicator should be emptied of unnecessary entries, as space is consumed and global names may be shielded by local labels, variables or arguments of functions in the state indicator.

The suspended function at the top of the state indicator may be restarted by entering →*N*, where *N* is a line number. The suspended function and any pending on it may be aborted by entering →. Response is a line showing the next suspended function if any.

```
      →
RUN[4]*
      )SI
RUN[4]*
TEST[6]*
```

)RESET state indicator reset

The entire run state can be cleared using)RESET. The resulting state indicator is reset:

```
)RESET
)SI
```

)*ATTACH*
)*DETACH*
)*GRP*

GROUP COMMANDS.

A group of names can be formed and named for collective reference including)ERASE or)COPY.

)ATTACH Groupname Nameset group association

The Groupname is the referent for the group. The Nameset provides the names that are associated with the group, and thereby, with each other. Normally, names in a Nameset match names of variables, functions or other groups. Names in the Nameset need not have any current meaning.

If Nameset is not present, the effect is to reserve Groupname, as a group, for subsequent attachment of a nameset. If the group Groupname already exists, the effect is to unite Nameset with the nameset already associated with Groupname (no name will be duplicated).

A group name included in Nameset causes the elements of that group's nameset to be implicitly included in the group.

If the Groupname is included in its own Nameset, then actions on the group apply also to the Groupname.

)ATTACH GROUP1 FNAME VNAME GROUP1
)ATTACH GROUP2 GROUP1 GROUP2 HOW

)DETACH Groupname Nameset group disassociation

The names in Nameset are detached from the group Groupname. If Nameset is absent, then the group Groupname ceases to exist.

Detach doesn't affect the existence of the names (other than Groupname). This is contrasted with)ERASE which eliminates the named objects.

)DETACH GROUP2

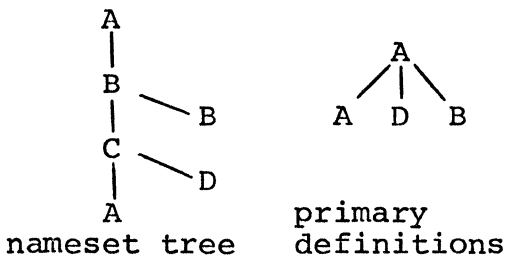
)GRP Groupname display group association

The names directly attached to Groupname are displayed in the order they were attached.

A group can contain in its Nameset its own name. If so, an action on the group nameset affects the group as well. A group (say G) can contain names of other groups. If so, an action on group G will replace each named group in its Nameset by that group's nameset. Any one group will only be replaced once. A second occurrence of a group name means the name itself rather than replacement. More than two occurrences of a group are ignored. The primary definitions of names in a Nameset are the unique names after substituting for first occurrence of any Groupname its Nameset, and retaining the Groupname on its second occurence.

```
      )CLEAR
CLEAR WS
      )ATTACH  A  B
      )ATTACH  B  B  C
      )ATTACH  C  A  D
      )GRPS
A B C
      )ERASE  A
NOT D
      )GRPS
C
      )GRPC
A D
```

The illustrations at the right show the nameset tree for group A after substitution of group namesets; and the resulting primary definitions. Note that the primary definitions include groups A and B, and undefined name D.



SECTION 6

SYSTEM VARIABLES AND SYSTEM FUNCTIONS

GENERAL.

The system variables provided within each workspace of the APL processor specially tailor the processing to the application of that workspace.

The system functions are provided to permit the user to perform many functions that query or alter the run environment of the account or to query the total environment of the APL system.

The classes of system functions include:

- Function transformations
- Name functions
- Debugging aids
- Execution controls
- Special character sets
- Status inquiries
- I-bar primitive functions

SYSTEM VARIABLES.

System variables are provided in a workspace and used by the APL processor to specialize its behavior for that workspace. These variables may be made local to a defined function. Values in limited ranges may be assigned to these variables.

System Variable	Name	Purpose	Default Value in clear WS	Range for n
⎕*CT* n	Comparison Tolerance	relative tolerance used in comparison with the primitive functions: ∩∪⊂⊃⌈⌊<≤=≥>≠∈⍳ (*DYADIC*)	1*E*¯10	0 *THRU* 1
⎕*IO* n	Index Origin	Origin for ordinal counting applies to the primitive functions: ⍳⍋⍒?[]⌽	1	0 *OR* 1
⎕*PP* n	Print Precision	The precision with which fractional or scientific notation numbers are rounded and printed, or formatted with monadic ⍕	10	1 *THRU* 12 *INTEGER*
⎕*RL* n	Random Link	Starting value for random number generator	131131704506	0 *TO* ¯1+2*39

The comparison tolerance is a relative tolerance used in comparisons. It helps resolve the problem of the finite precision with which numbers are represented within the computer. In a dyadic function the comparison tolerance is relative to the left argument. For example:

```
A=B←→⎕CT≥|(A-B)÷A
A<B←→⎕CT≥(B-A)÷A
```

The comparison tolerance is also used for domain checking where the domain of the function is non-continuous, e.g., integer or Boolean domain.

The index origin affects the denumeration of elements and the coordinates in an array.

origin	denumeration begins with
0	0
1	1

The index origin affects the following:

⍉	permute (dyadic left argument)
⍳	indices (monadic), index (dyadic)
⍋⍒	grade up, grade down
?	roll (monadic), deal (dyadic)
	subscripts on arrays [bracketed]
	coordinate selector [bracketed]
	laminator [bracketed]
	file component selector [bracketed]

The print precision affects the result of all numeric outputs in fractional or exponential form. No more than ⎕PP significant digits are displayed. Rounding is envoked first. Integers are displayed with full precision so long as their magnitude is less than (2*39)-1. Also, print precision affects the character object result of default formatting using ⍕.

The random link affects the result of the roll and deal functions. The random link is used as the seed to the random number generator. Each time the random number generator is used the seed is used to determine the next value(s) delivered. Each use delivers a result and changes the seed. Given the same seed and same range, the random number generator will generate the same random numbers (and return the same new seed).

FUNCTION TRANSFORMATIONS.

System Function	Name	Result	Remarks
□*CR N*	Canonic Representation	Character matrix. Each row is line of function N. First row is function header. Line numbers and opening and closing dels are omitted.	N is name of unlocked defined function. If not, result has shape 0 0.
□*VR N*	Vector Representation	Character vector with each line of function N terminated by return character after the last non-blank.	N is name of unlocked defined function. If not, result is empty vector.
□*FX C*	Fix	Defined function. Function name will be that from first line of C. If that name is local to function, the fixed function is also local. If an explicit result is required, it is the name of the function.	C is either character vector or matrix in form from canonic or vector representation.

Canonic Representation of a function is useful for user-written function editing routines where line rearrangement, function merging or separation is desired. Note that the shape of the result is number of lines (including header) by length of longest line. Thus this form generally takes more space than vector representation, particulary if the line lenghts differ.

The Vector Representation is usually the more compact representation, and is the preferred form for storing functions as file components.

A Fix of a character representation returns the function in unexecuted form. This form takes the same space as a)COPY; slightly more than after first execution.

If the defined function name resulting from a Fix is local to some function in the calling sequence resulting in executing the Fix, then the fixed function is local to that function. Of course, that name at the time of the Fix must not be assigned a meaning. The definition of the fixed function disappears upon exit of the function to which the fixed function is local.

NAME FUNCTIONS.

Name system functions work with a string or matrix of names.

System Function	Name	Result	Remarks
⎕NL N	Name List	Matrix of names of objects of specified kinds in the current environment. Names are alphabetized, left justified, one per row.	N is numeric scalar or vector selecting object kinds: 1 labels 2 variables 3 functions 4 other (groups)
A⎕NL N	Selective Name List	Like Name List except only includes names starting with a character in String A.	A is chosen from letters, underscored letters, Δ and Δ
⎕NC C	Name Classification	Vector of integers indicating in the current environment for corresponding name in character string or matrix C.	Result value meanings: 0 no associated object 1 label 2 variable 3 function 4 other (group)
⎕EX C	Expunge	If required, the result is a Boolean vector with ones everyplace the corresponding name from C was expunged.	Objects corresponding to names in character vector or matrix C are expunged. The objects must not be labels, groups or executing, suspended or pendent functions.

A character string argument to name classification or expunge must contain only one name. A character array argument must contain one name per row.

The most local occurrence of a name in the current environment determines its kind. More global occurrences may be shielded by a local occurrence as a local name or label in a function currently in execution, pending or suspended in the state indicator. The more global meaning returns upon exit from the function to which the name is local.

Expunge may be used to eliminate current meanings for objects from the current environment so long as they are not labels or functions currently in execution, pending or suspended. Unlike)ERASE, local variables or functions not in the state indicator can be expunged.

DEBUGGING AIDS.

The following system functions are oriented to lines of unlocked user-defined functions.

Monadic All lines		Name	Dyadic Specified Lines		Result
□*ST*	F	Set Trace	N	□*ST* F	L
□*SS*	F	Set Stop	N	□*SS* F	L
□*SM*	F	Set Monitor	N	□*SS* F	L
□*RT*	F	Reset Trace	N	□*RT* F	L
□*RS*	F	Reset Stop	N	□*RS* F	L
□*RM*	F	Reset Monitor	N	□*RM* F	L
□*QT*	F	Query Trace			B
□*QS*	F	Query Stop			B
□*QM*	F	Query Monitor			B
□*MV*	F	Monitor Values	N	□*MV* F	V

Where

F is character vector name of unlocked user-defined function
N is numeric vector of line numbers
L is numeric vector of lines with property (set, reset); this result is returned only if required.
B is Boolean vector, 1 if property set, 0 if reset one element per line including header.
V is vector of numeric monitored values accumulated during executions since set.

The monadic forms apply to all lines including the header line 0. The dyadic forms apply only to altering the current setting for line numbers in the left argument.

During function execution the effects of the aids are as follows on encountering a line in which set:

	header line 0	body line
Trace	result returned by function	result
Stop	suspend prior to return	suspend prior to execution
Monitor	increment number of calls	increment CPU time in line execution

The trace result forms are:

```
Function-name [Line-number]

Function-name [Line-number]   Type (Shape)  Value
```

The first form occurs if no result is possible; otherwise, the second form occurs.

The type is C for character or N for numeric. The Shape is a numeric vector; the Value is the normal displayed value.

The stop result form is:

```
Function-name [Line-number]*
```

The monitor values are internally accumulated to more precision than they are displayed. The ceiling of the accumulated number of milliseconds is displayed. A time of 0 is shown only for unmonitored lines or monitored lines that have not been executed. Thus, monitoring all lines over a period of execution is an effective way to determine if some program path has reached each line, and also the time spent in each line.

If a line contains a call on another function, any time spent in that called function would be accumulated there, instead of in the calling line.

EXECUTION CONTROLS.

The normal execution can be altered using the following system functions.

System Function	Name	Result	Remarks
□*DL D*	Delay	Optional actual delay D	in seconds
□*ED S*	Edit	edited line after editing	uses normal line editing on string S
B □*ED S*	Phrase Edit	edited line after editing string S according to Boolean vector B	ones in B are phrase terminators
□*ER S*	Error	error message S displayed, no other explicit result	use in locked function

The specified Delay amount D is an integer indicating minimum desired execution pause before resumption. The actual delay, returned if required, also includes time awaiting an APL processor once the specified delay has occurred.

Each Edit function accepts a character string as right argument. This string may not include any of the following characters: linefeed, return, backspace, tab or null. The monadic form displays the string and returns to the left margin for entry of a line of edit characters applied to the characters above: ' ' for delete, '.' for phrase end before, and spaces for no change. The next line displays the first phrase for editing. The ATTN causes entry of the next phrase, etc. The dyadic form uses the Boolean left argument (of same length as the string) with each one indicating a phrase end. This avoids the line of entered edit characters.

The Error message is displayed, an error indication prompt is given, and execution is suspended. This is principally useful in a locked function. In that case, the error message results in the suspension point indicator being in the line of the calling function containing the call, rather than in the line containing the error message.

SPECIAL CHARACTER SETS.

The special character sets below are niladic value returning system functions.

System Function	Name	Result	Remarks
⎕*B*	Backspace	scalar backspace character	in same line
⎕*L*	Linefeed	scalar linefeed character	no return
⎕*R*	Return	scalar carrier return character	includes line feed
⎕*T*	Tab	scalar tab character	to next physical tab
⎕*N*	Null	scalar null character	no effect on display
⎕*A*	Alphabet	character vector 'ABC...Z'	
⎕*D*	Digits	character vector,'0123456789'	

These characters are processed internally to APL just as any other elements of a character data object. The only special properties of the first five are associated with output processing for terminal display.

The Backspace character can be used to create overstruck output characters not in the allowed character set. It can not be used to move to the left of the start of the display line.

The Return character causes completion of an output line, just as the RETN key does for input. It includes both line feed and cursor return to the left margin.

The Linefeed character can be used for advancing the display line while the cursor is positioned into a line with return.

In cases where the cursor is at the left margin, the Linefeed and Return have the same external effect.

The Tab character can be used to prepare output with irregular terminal physical tab settings. In this use, the normal APL editing to insert tabs in output for display should be disabled. The tab interval should be set to 0 by)TABS 0.

The Null character takes one unit of transmission time when sent to the display, but has no visual effect on the normal static display. Its principal use will be with non-standard display devices such as plotters that may require time to complete a prior command.

STATUS INQUIRIES.

The status inquiries below are niladic value returning system functions.

System Function	Name	Result	Remarks
□*PT*	Print tabs	uniform tab interval assumed for terminal	set by)Tabs n
□*PW*	Print width	maximum characters per display line	set by)Width n
□*WI*	Workspace ID	character vector: workspace identifier	)WSID
□*AN*	Account name	character vector: account identifier	⍳29←→□*AN*
□*AI*	Accounting Information	computer time, connect time this session	in milliseconds
□*LC*	Line count	numeric vector: includes line on which line count occurs, then other line numbers of functions in the state indicator	⍳27←→□*LC*
□*TS*	Time Stamp	numeric vector: year, month, day, hour, minute, second, millisecond	Example 1974 12 31 23 59 59 999
□*UL*	User load	number of user accounts on APL	⍳23←→□*UL*
□*WA*	Working area	bytes remaining, bytes in use in workspace	⍳22←→1↑□*WA*
□*NA*	Name area	slots remaining, slots assigned in name table	⍳31,⍳30←→□*NA*
□*LA*	Library area	workspace slots remaining, workspaces in)LIB	⍳33←→1↑□*LA*
□*FA*	File area	File slots remaining, files in)FILES	

The use of the above status inquiries are generally preferable to the related I-bar primitives. The sum reductions of the last two area inquiries provides the quotas established by the installation for the account. The sum reduction of Name Area is the number of symbols in the name table, set by)SYMS n for the clear workspace default, or)CLEAR n for a particular workspace.

I-BAR PRIMITIVE FUNCTIONS.

The set of monadic primitive functions defined in early APL systems for querying the environment have the form:

⌶N

where N is an integer between 20 and 33, excluding 28.

These primitives are redundant, having been replaced by the system functions. Since they may exist in old APL programs, they are described here. Note that replacements may have different units, or may be vector instead of scalar.

Time units below are sixtieths of a second for I-bar results:

Primitive	Result	Approximate Replacement
⌶20	Scalar time of day	3↓⎕*TS*
⌶21	Scalar CPU time used this session	1↑⎕*AI*
⌶22	Scalar bytes remaining unused in the workspace	1↑⎕*WA*
⌶23	Scalar number of users currently signed on	⎕*UL*
⌶24	Scalar time of day at start of the work session	1↓⎕*AI*
⌶25	Scalar date in form MMDDYY where M,D,Y are digits representing month, day, and year respectively	100⊥100|1⌽3↑⎕*TS*
⌶26	Scalar first element of I27	1↑⎕*LC*
⌶27	Vector of line numbers in state indicator. First element is line being executed, or the one last suspended. The next element is the line which called the first, or the prior suspension, etc.	⎕*LC*
⌶29	Character vector containing account identification.	⎕*AN*
⌶30	Scalar positions used in name table	1↓⎕*NA*
⌶31	Scalar positions available in name table	1↑⎕*NA*
⌶32	Scalar CPU time remaining before use quota is exhausted; if 0, then no quota exists.	
⌶33	Scalar workspaces remaining unused	1↑⎕*LA*

Note that there is no I-bar 28, keyboard unlocked time, implemented on some other APL implementations. This time is not available in APL/700 since the actual times for keyboard unlocking (after completion of transaction processing and any display information) and for acceptance of data from a terminal are queued and not known to the APL processor. A high approximation to this time is the difference between the connect time and CPU time this session. It should be reduced by the time for transmitting display information, and the queueing time awaiting an APL processor. Neither of these times are available in APL/700.

SECTION 7

FILE SYSTEM OPERATORS

GENERAL.

The APL/700 System includes a filing system which provides users with effective and convenient means to retain and access APL data objects outside the workspace. Defined functions can be represented as data objects and subsequently can be fixed back into the functions. Thus, a user can work with more data or functions than will fit in a workspace at one time.

FILE NAME.

Each file has a name unique among the file names of the account.

File Name is (Acct) Name [Password]

where File Name and optional Password are strings of 1 to 12 alphanumeric characters starting with a letter.

The optional Acct is the account name required if the file is owned by another account. The Acct is a string of 1 to 6 alphanumeric characters.

FILE COMPONENTS.

At any time a file has a number of components. These are numbered starting with the index origin. Any component may be null, or may contain a value. A component can contain any APL data object created in a workspace and subsequently assigned to the file component. Each component is independent, and can have any type, rank or size. In particular, some components can be user created directories to the file. A null component is one that has no value (this is different from containing an empty vector as a value).

FILE LIMITS.

Any file has a maximum of 1000 component slots. The installation allocates to an account a maximum number of files, and a maximum number of bytes per file. There is a system imposed maximum number of files that can be concurrently opened by any one user.

FILE OPENING, ACTIVE AND INACTIVE STATUS.

A file may be open in one or more accounts. A file has active status if any account has the file open, otherwise, the file is inactive.

A file is opened for an account when first any file operation is executed other than create, rename, destroy, or file existence test. A file remains open until either explicit release, or account sign-off.

NOTES.

The file system operators for APL version 2.6 are being extended in utility for version 2.7.

File updating integrity over system failure has been improved. The period over which an update transaction occurs may be extensive. Any transaction entries while the user has the file held are provisional. They become permanent only when a file free is executed as part of a user-defined function. Any return to calculator mode before the free occurs removes the provisional transaction. This capability protects the file from being partially updated.

Some file operators return the file name if required. This permits a sequence of file operations to be executed in the same line of a defined function.

Additional operators are provided including file rename, file reverse, file compress, file expand, file release, and some additional queries.

One restriction is added. A file can not contain more than 1000 components.

FILE SYSTEM PRIMITIVE OPERATORS.

A group of primitive operators is provided for file management. Each file operator is denoted by overstriking the quad (box) symbol with another symbol. The resulting operator has generally similar meaning to the APL primitive functions using the same second symbol.

Many of the file operators have both monadic and dyadic forms. The right argument of each is the File Name, symbolically represented as 'F'.

FILE CREATE, CHANGE PASSWORD, RENAME AND DESTROY.

A file can be created or destroyed, and its password or name can be changed.

Δ *CREATE FILE*
Δ *CHANGE PASSWORD*
Δ *RENAME FILE*
∇ *DESTROY FILE*

Forms:

	Δ *F*	Create file
	Δ *[O/N] F*	Change password on file
G	Δ *F*	Rename file to become G.
	∇ *F*	Destroy file

Where: F is own account File Name, may include password
G is new File Name for own account
O is old password for file F, empty if none previously
N is new password for file F, empty if none desired

Actions/Results:

Create:	Creates file with name File Name and no components. If required, the file name is returned. Does not open file.
Change Password:	Changes password on existing file. If required, the file name is returned.
Rename File:	Renames file F to become G. Does not open file.
Destroy:	The file File Name owned by this account is destroyed if it exists and is not currently held by any other user. If required, returns 1 if successful, 0 otherwise.

Conditions/Options:

Create:	File name must not already exist in account.
Change Password:	Only the file owner can change the password. This can only be done when the file is not held by another account. Add password if O is empty. Change password if O and N are not empty. Delete password if N is empty.
Rename File:	A file can only be renamed if inactive.
Destroy:	The File Name including lock if any must be provided. A file of another account cannot be destroyed.

Examples:

Δ'*NEWFILENAME*'
Δ'*LOCKEDFILE*[*KEY*]'
Δ'*NEWFILENAME*[*/KEY*1]'
∇'*LOCKEDFILE*[*KEY*]'

```
⍈ NULL COMPONENT
⍈ WRITE COMPONENT
⍇ READ COMPONENT
```

FILE COMPONENT NULL, WRITE AND READ.

Specific file components can be set null, written into or read.

Forms:

```
    ⍈ [K] F    NULL component K of file F
A   ⍈ [K] F    WRITE A to component K of file F
    ⍇ [K] F    READ component K of file F
```

Where: F is File Name
K is component number
A is any APL data object

Actions/Results:

NULL: Destroys prior value of component K.
If required, returns File Name.

WRITE: Replaces prior value of component K by value A.
If required, returns File Name.

READ: Returns the non-null value of component K.

Conditions/Options:

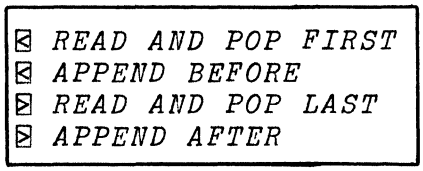
NULL: K must be an existing component number.

WRITE: K must be either an existing component number or one greater than the prior last component number. In the latter case, an append to the end is done instead. (This append is not available in 2.6 release).

READ: The component must be non-null.

Examples:

```
      ⍈[3]'FILENAME'
      2 5⍈[2]'FILENAME'
      'SMITH'⍈[3]'FILENAME'
      ⍇[2]'FILENAME'
2 5   ⍇[3]'FILENAME'
SMITH
```



FILE COMPONENT POP AND APPEND.

The file components may be treated as a stack or a queue. A component may be appended to either end. The component at either end may be read and removed (popped).

Forms:

	⍁ *F*	Read and pop first component of File
	⍂ *F*	Read and pop last component of File
A	⍁ *F*	Append component before components of File
A	⍂ *F*	Append component after components of File

Where: F is File Name
A is any APL data object

Actions/Results:

Pop: The result returned is the indicated first (last) component. That component must be non-null. The popped component is removed from the file. If first, the component numbers of the old components are decreased by 1.

Append: The data object is appended before (after) the existing file components. If before, the component numbers of the old components are increased by 1.

If required, the file name is returned.

Examples:

```
      'JONES'⍁ 'PERSONS'
      'SMITH'⍂'PERSONS'
      (2 2 ⍴ 1 1 4 7) ⍁'FILENAME'
      ⍁ 'PERSONS'
JONES
      ⍁'FILENAME'
1 1
4 7
```

```
⍉ REVERSE ORDER
⍉ ROTATE CIRCULAR
```

FILE COMPONENT REVERSE AND ROTATE.

The component order may be reversed or circularly rotated. Like the primitive reverse and rotate functions.

Form:

	⍉ *F*	Reverse component order in file
I	⍉ *F*	Rotate circularly the components in file

Where: F is File Name
I is integer

Actions/Results:

Reverse: The component order of File F is reversed; i.e., the first changes with the last, the second changes with the second last, etc. If required, the File Name is returned.

Rotate: The components of file F are rotated circularly by an amount I. If I is negative, this is effectively a right rotate. If required, the File Name is returned.

Conditions/Options:

Reverse: Not available in 2.6 release.

Rotate: I is effectively the (number of components) residue of I. I=1 causes the first component to become the last, the second component to become the first, etc.

Examples:

```
⍉'FILENAME'
2⍉'FILE[LOCK]'
¯3⍉'FILENAME'
```

↑ *TAKE COMPONENTS* ↓ *DROP COMPONENTS*

FILE COMPONENT TAKE AND DROP.

The remaining file components may be the result of taking or dropping components from either end. These are like the primitive take and drop except that they are destructive of components dropped or not taken.

Form:

I ↑ F	Take I components from File
I ↓ F	Drop I components from File

Where: F is File Name
I is integer magnitude≤ 1000
I>0 applies to components from start of file
I<0 applies to components from end of file

Actions/Results:

Take: The resulting file F has I components.
If required, the file name is returned.

Drop: The resulting file F has I components dropped.
If required, the file name is returned.

Conditions/Options:

Take: If the magnitude of I exceeds the number of components previously in the file, sufficient null components are appended to the file at the appropriate end:

before if I<0
after if I>0

Drop: A minimum of 0 components remain.

Examples:

```
5↑'FILENAME'
¯23↑'FILENAME'
2↓'FILENAME'
```

⍁ *COMPRESS COMPONENTS*
⍂ *EXPAND COMPONENTS*

FILE COMPONENT COMPRESS AND EXPAND.

The ordered set of file components can be expanded or compressed. These operators are similar to the primitive expand and compress functions.

Forms:

B ⍁ F	Compress components of File
B ⍂ F	Expand components of File

Where: F is File Name
B is Boolean vector

Actions/Results:

Compress: The result is an ordered component set selected in order from the components previously in F, wherever a 1 exists in the Boolean B.
The components of the original file are destroyed wherever a 0 exists in B.
If required, the file name is returned.

Expand: The result is an expanded, ordered component set preserving the order of the original components within which null components are inserted wherever zeros exist in Boolean B.
If required, the file name is returned.

Conditions/Options:

Compress: The length of B must be the same as the number of components in the original file F. $(\rho B)=3\boxdot F$

Expand: The number of ones in B must be the same as the number of components in the original file F. $(+/B)=3\boxdot F$

Examples:

```
1 1 0 1 ⍁'FILENAME'
1 0 1 0 1 ⍂'FILENAME'
```

⍟ *FILE EXISTENCE*
⍟ *QUERY FILE ATTRIBUTE*

FILE EXISTENCE AND QUERY.

The existence and attributes of a file can be determined.

Forms:

	⍟ F	Test existence of file
I	⍟ F	Query attribute of file

Where: F is File Name
I is integer

Actions/Results:

Existence: Result is Boolean: 1 if file named F exists, 0 otherwise. Does not open file.

Query:

I	Result
1	Current size of file in bytes.
2	Maximum size of file in bytes.
3	Number of components in file, including nulls.
4*	Boolean, 1 if any modification since file was last organized.
5*	Number of accounts with file open.
6*	Cycle number of last reorganization.
7*	Last update time stamp.

Conditions/Options:

Query: The options marked with * are not implemented in 2.6 release.

2	Maximum size of file is an installation option.
3	Maximum number of components is 1000.
5	Whenever the number of accounts with file open goes to 0, the file is reorganized if it has been modified.
6	Reorganization causes merging of the update file into the main file. The reorganized file is compact and in indexed sequential form. The file becomes temporarily unavailable during reorganization. Reorganization is initiated any time the file becomes inactive.
7	The time stamp is a 7 element vector - year, month day, hour, minute, second, millisecond - indicating time of last modification to the file.

Examples:

```
      ⍟'FILENAME'
1
      ⍟'NOSUCH'
0
      3⍟'FILENAME'
14
```

⌸ *COMPONENT MAP*

FILE COMPONENT MAP.

The components of a file that are null and non-null can be determined.

Form:

⌸ F Determine map of file

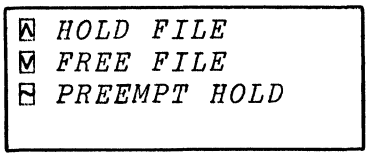
Where: F is File Name

Actions/Results:

The result is a Boolean vector with length the number of components. In component order, the resulting element is 0 if the corresponding component is null; 1 if the corresponding component is non-null.

Example:

```
      ⍋'FILE'
      1 2 3 4 ⍒'FILE'
      3 ⍓'FILE'
      ⌸'FILE'
1 0 0
```



FILE HOLD, FREE, PREEMPT.

In file use shared among several accounts, exclusive use can be achieved for critical up-dates.

Forms:

HF	Hold file for exclusive use
FF	Free hold on file
PF	Premptively hold file

Where: F is File Name

Actions/Results:

Hold: If the file is not currently being held, a hold is placed on the file which prevents any other account from accessing it.
If already held by another account, hold causes a wait until freed.
If required, the file name is returned.

Free: A held file is freed from exclusive use.
If required, the file name is returned.

Preempt: The account owning the file can preemptively break an existing hold by some other account and place its own hold on the file.
This causes any up-date in progress by the other account to be discarded.
If required, the file name is returned.

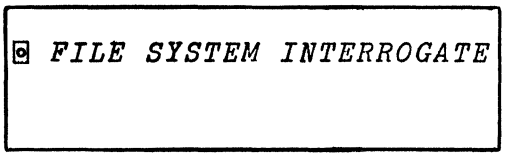
Conditions/ Options:

Hold: A hold only persists while execution continues in a defined function. Any return to calculator mode (or file destroy while held) breaks the hold.

Free: In version 2.6, actual file up-dates take place as indicated by the file operations.
In version 2.7, the actual file up-dates take place provisionally into the up-date file.
They are accepted as up-dates when the free occurs.
Any interruption before the free voids the provisional entries.

Examples:

H'(*OTHER*)*FILE*'
F'(*OTHER*)*FILE*'
P'*FILE*'



FILE SYSTEM INTERROGATE.

Usage properties of the file system can be determined.

Form:

I	Interrogate file system
Where:	I is integer

Actions/Results:

I

1	Returns current number of accounts using files.
2	Returns current total number of files that are active.

Conditions/Options:

Neither number may exceed installation set maxima.

Examples:

```
1
5
2
11
```

SECTION 8

ERROR REPORTS AND INTERPRETATION

GENERAL.

The APL/700 System has a comprehensive error-reporting capability that aids users in determining the cause of errors and resultant corrective action. This capability is one of the advantages of the conversational, interactive APL/700 environment in that it enables the user to experiment very easily by the trial-and-error method. However, the user should be careful not to incorrectly generalize or misinterpret the results of his experimentation.

This section provides descriptions of the various types and forms of errors and provides sufficient background information to aid the user in experimentation or interpreting and correcting errors. A complete listing of APL/700 error reports is contained in Table 8-1.

ERROR REPORT FORMATS.

When an instruction/command is entered into the APL/700 System, the computer attempts to execute it. If the computer cannot complete the execution process required, it stops and returns an error message to the user terminal. Each error message consists of up to three output lines, as applicable. The first line is always a typed report identifying the error in the following format:

```
***ERROR REPORT***
```

If an error occurs during a definition mode, the message also contains a second line which restates the instruction in the form that the system reads it. The third line contains a caret symbol (∧) marking the point in the instruction at which the operation encountered trouble and could not continue. For example, if the length of a vector argument is incorrectly stated for an operation using one or more vector arguments, the entry and error message format is as follows:

```
      8 6 7+5 3

***LENGTH ERROR***

      8 6 7+5 3
          ∧
```

The one variation to the above format is in the case of the "CHARACTER ERROR" report, where an invalid overstrike was typed to form an invalid character (such as overstriking x with ÷). The squat quad (⍞) symbol

("[" overstruck with "]") is used instead of the caret to mark this error.

The "Report" column of Table 8-1 lists, in alphabetical order, all of the possible error reports provided by APL/700. Each report is displayed in the exact form shown in the column, except that it is suffixed and prefixed by asterisks (stars) as noted previously. The "Definition" column of Table 8-1 lists the system interpretation of the cause for each report. Where applicable, relevant statements are provided to aid in corrective action.

TYPES AND FORMS OF ERRORS.

There are several types of errors comprising the complement of error reports: the limit type, the file type, and the true error type. A limit type error occurs when the user tries to do something which exceeds the capacity of the computer or the capability of APL/700. For example, a "SPACE LIMIT" report occurs when the user attempts to use more space than is available in his active workspace. As shown in Table 8-1, a large number of possible errors relate to file system operations. For example, a "FILE USERS LIMIT" report occurs when the maximum number of file users are currently using the file system and an additional request cannot be accepted. A "FILE LOCKED" report is returned when the user enters an incorrect lock or no lock in a locked file reference.

Many errors may result from true user or transmission errors, such as syntax or context errors. These may be caused by a user typing mistake or a system malfunction. When an instruction is entered by a user, first the computer has to read it, then it has to execute it. Two types of errors may cause the computer to read the instruction incorrectly, or not read it at all. One is a transmission error, which may be caused by electrical faults or noisy transmission lines. The other is a character error, which may arise when the input doesn't refer to an allowable APL character, even though the transmission is technically adequate.

User errors are caused by various reasons. The user may misunderstand the proper use of an operation, try to carry out a sequence of instructions in the wrong order, or have forgotten what value is associated with a variable. A great many errors are simply mistypings. The computer, of course has no way of knowing what the user intended; it executes the instructions, to the best of its ability, until it encounters something that it cannot execute. Then it reports the trouble that it has encountered. The system classification of the error is the system interpretation of the error; it cannot guess how the error departs from what the user intended. For example, if the user misspells the name of a variable, the computer may read this as a reference to some other variable, and it will report an error only if the value of that other variable makes the instruction impossible to execute. Thus, the computer can't stop and report the spelling error, even if it is the true cause of the error.

Similarly, if the user types a parenthesis in the wrong location, or omits a required entry, the computer can only report what problem it encountered as it tried to execute the instruction. Thus, while the computer reports the type of error it has found, it can't tell the user what he should have typed. This has to be determined by the user alone.

Normally, when the computer finds an error in an instruction, the instruction has to be edited or reentered. The value of an intermediate expression within the instruction is not saved, unless the instruction specifically directs that it should be stored as the value of a named variable. This arises only when there is a specification arrow further to the right (and hence executed earlier) than the caret that indicates where the trouble is. If the result of an intermediate step has been stored, only the part of the instruction that appears to its left has to be reentered.

The following paragraphs elaborate on some of the more common errors that may be encountered.

SYNTAX ERROR.

When the user enters an expression whose syntax is invalid, the "SYNTAX ERROR" message is reported. Some examples of invalid syntax entries are as follows:

a. Two variable names are juxtaposed (placed side-by-side) with no indication of the operation that is to be performed on them.

b. An operator symbol is used with no indication of a value on which it is to operate.

c. A parenthesis or bracket is opened but not closed, or closed but not opened.

d. A defined function is used in a way that is inconsistent with the syntax specified in its header.

DEFINITION ERROR.

When the user enters an instruction employing the del (∇) symbol improperly, a "DEFINITION ERROR" message is reported. Some of the incorrect usages of the definition mode are as follows:

a. The del (∇) symbol is not the first character in the instruction, nor within quotes.

b. An attempt was made to reopen the definition of a function whose name appears in the state indicator other than on top, or to alter the header line of the suspended function on top of the state indicator. In this case, check the state indicator by entering ")SI". Reset the state indicator by entering ")RESET".

c. An attempt was made to start a new definition for a function whose header contains a result, an argument, or a local variable when a definition for a function of that name exists in the workspace.

d. While in the definition mode, a defective request was entered to edit a line of the function.

DOMAIN ERROR.

When an instruction entry asks an APL operator to operate on a value outside the domain that the operator can handle, a "DOMAIN ERROR" message is reported. A domain error will also occur if an attempt is made to divide by zero.

TYPE ERROR.

A "TYPE ERROR" message will be reported if the type of entry is incorrect for the operation being performed. That is, if an attempt is made to do arithmetic on a value which is not a number, or to catenate a literal character object with a numeric object, or to insert character elements into a numeric array, or to insert numeric elements into a character array.

VALUE ERROR.

When a "VALUE ERROR" message is reported, it indicates that the user-entered instruction refers to a name for which no value can be found in the designated workspace. This may arise because the user failed to assign a value to that name to make it a variable, or because he misspelled the name so that the computer does not recognize it, or failed to define a function of that name. In this case, the situation may be corrected by entering a value for the missing variable, or correcting the misspelled name.

Value errors may also arise if an attempt is made to make use of the result of a defined function, but the function definition fails to provide one. This can be remedied by rewriting the function definition so as to provide an explicit result, or (if it already has one) by making sure that the body of the definition in fact specifies a value for the result before execution of the function is complete.

RANK ERROR.

The rank of a variable is the number of dimensions it has. A "RANK ERROR" message is reported if an entered instruction uses variables of different rank for a function which requires that the ranks be matched, or a variable whose rank is too large for the particular function. While the scalar functions extend to arrays of any rank, a number of the other functions, such as monadic ⍳, or the left argument of ⌽, can take arguments only of rank 1 or rank 0.

Table 8-1

Error Reports

Report	Definition
ACCOUNT ACTIVE	An attempt was made to sign on an account that is already signed on to APL.
ACCT-NAME ERROR	A reference was made to a nonexistent account, or the name was improperly formed.
BUFFER LIMIT	An attempt was made to execute a string longer than the buffer, or an attempt was made to set the prompt to be a string longer than the buffer. The buffer length is 130 characters.
CHARACTER ERROR	An invalid overstrike was entered. The locations of the invalid overstrikes are indicated by the squat quad (⍞) symbol.
CONTEXT ERROR	A name was used out of context with its current definition.
CONTROL ERROR	A parameter to a command was incorrect.
DEFINITION ERROR	An attempt was made to define a new function with a name that already exists, or the function header was improperly formed. (Refer to Section 4.)
DIMENSION ERROR	The dimension specified does not exist. (This occurs with a function which operates on one of several dimensions.)
DOMAIN ERROR	The argument of a function was outside the range of acceptable values for that argument to the function.
DUP-NAME ERROR	An attempt was made to give a local name multiple definitions of different classifications.
EDIT ERROR	Something other than a ' ', '/', or '.' editing control symbol was typed beneath a line when in the line edit mode.
FILE ACTIVE LIMIT	The user has the maximum number of active files permitted; no more requests to make more files active can be accepted.
FILE ALREADY EXISTS	An attempt was made to create a file that already exists.

Table 8-1 (cont)

Error Reports

Report	Definition
FILE ERROR	Execution of APL was halted, or a line-drop occurred while a file operation was in process.
FILE INDEX ERROR	An attempt was made to read or write a component of a file that does not exist in the file.
FILE LOCKED	No password, or an incorrect password, was used in a file reference.
FILE-NAME ERROR	An attempt was made to use an improperly formed name as a file name.
FILE NONCE ERROR	The file operation referenced is not presently implemented.
FILE NONEXISTENT	The referenced file does not exist.
FILE QUOTA LIMIT	An attempt was made to create more files than the account is permitted.
FILE SPACE LIMIT	The area reserved for the file has been exhausted.
FILE SYSTEM ERROR	An unexpected execution error occurred in the file system. (This should be reported to the system manager; all relevant output should be saved.)
FILE SYSTEM LIMIT	The maximum number of files allowed to be active are currently active; no more requests that activate a new file can be accepted at present.
FILE UNAVAILABLE	The referenced file is unavailable at this time.
FILE USERS LIMIT	The maximum allowable number of file users are currently using the file system; no more file users can be accepted at this time.
FILE VALUE ERROR	An attempt was made to access a null component of a file. The file operation was completed.

Table 8-1 (cont)

Error Reports

Report	Definition
FORMAT ERROR	The left argument to the format operator is not a valid format.
GRP-NAME ERROR	A reference was made to a nonexistent group.
INDEX ERROR	An index into an array was out of the array bounds.
INTEGER LIMIT	A number larger than the largest integer that may be represented by the machine was used where an integer was needed. (The magnitude of the largest integer is 549755813887, or ¯1+2*39.)
INTERRUPT ERROR	An error was forced at a non-suspendable point by striking the attention key twice.
LENGTH ERROR	The length of a vector is incorrect for an operation using one or more vector arguments.
NAME ERROR	An argument to a system function requiring a name was given an improperly formed name, or a name with incorrect meaning was given.
NONCE ERROR	An attempt was made to use a feature that is not presently implemented.
NO SHARES	The shared variable facility is not available at this time.
NUMBER LIMIT	The result of a computation is a number greater than the largest number that the machine can represent. (The magnitude of this number is 4.31359146674E68.)
PASSWORD ERROR	An incorrect password was used.
RANK ERROR	The rank of an object is incorrect for the operation being done.
RANK LIMIT	An attempt was made to create a structure whose rank was greater than the maximum allowable. (Data structures may not exceed rank 16.)

Table 8-1 (cont)

Error Reports

Report	Definition
SHAPE ERROR	The shapes of objects are incompatible for the operation to be performed.
SHARE QUOTA LIMIT	An attempt was made to share more variables than the processor is permitted to share.
SHARE SPACE LIMIT	An attempt was made to use more shared variable space than the processor is permitted.
SIGN-ON ERROR	An incorrect sign-on entry was made.
SIZE ERROR	A one-element object was needed for an operation, but it was not found.
SPACE LIMIT	An attempt was made to use more space than is available in the active workspace.
STATE ERROR	A edit request was made on a function which would cause the state indicator to be incorrect if the edit were performed.
SYMBOLS LIMIT	An attempt was made to create more symbols than there is space for in the symbol table. (Unless otherwise specified by the user, there is space for 256 symbols.)
SYNTAX ERROR	The syntax of the APL expression entry is incorrect.
SYSTEM LIMIT	APL encountered an unexpected error during execution. (This problem should be reported to the system manager; all relevant output should be saved.)
TIME-QUOTA LIMIT	This error occurs once an account has exceeded its computer usage quota. The user session is then terminated, and the quota must be increased before the account may use APL again.
TYPE ERROR	The type of entry structure is incorrect for the operation being done.
WS-NAME ERROR	A reference was made to a nonexistent workspace, or the name was improperly formed.

Table 8-1 (cont)

Error Reports

Report	Definition
WS-QUOTA LIMIT	A ')SAVE' could not be executed because the account has used all available workspace slots. Some workspaces must be dropped, or the workspace quota for the account must be increased.
VALUE ERROR	An attempt was made to use name (variable) to which no value has been specified.

APPENDIX A

GLOSSARY OF TERMS, ABBREVIATIONS, AND ACRONYMS

The following is a glossary of terms, abbreviations, and acronyms used in APL/700.

Term, Abbreviation, or Acronym	Definition
Across (relative to datum dimension)	An orientation of a plane relative to the dimensions of a datum. Planes are said to be "across" a dimension when they are orthogonal (at right angles) to that dimension. A specific plane orthogonal to a dimension is sometimes referred to as being the i-th plane "across" that dimension, that is, the i-th plane encountered travelling "along" a vector parallel to the axis of that dimension.
Along (relative to datum dimension)	An orientation of a vector, relative to the dimensions of a datum. Vectors can be considered to be "along" a dimension when they are parallel to the axis of that dimension.
APL	A Programming Language. An interactive programming language for describing procedures in a time-sharing environment.
Argument	A datum (or list) supplied to a function or procedure.
Array	A datum having shape. An array may be a vector, a matrix, or an n-dimensional object and may have one or more elements or no elements.
Array Index	The index (number location) of an element in an array, relative to the origin (usually 0 or 1) of that array.
Array Origin	The sub-array selected by holding two or more indices at the origin value.
Calculator mode	Normal mode of APL/700 System, in which instructions are executed.
Comment (APL)	Any niladic text prefixed by the lamp symbol (⍝).

Appendix A

Glossary

Term, Abbreviation, or Acronym	Definition
Component	Any "element" of a list. An APL value is any character or numeric form (scalar, vector, matrix, or an element of any number of dimensions).
Coordinate	The entire set of plane indices for a particular point (in a matrix or array), ordered from the first to the last dimension.
Corner	Any n-dimensional sub-array having a number (n) of its faces that are sub-faces of an n-dimensional array.
Datum	A single data object. It may be a scalar or an array, but not a list.
Defined Function	A procedure or program made up of steps containing APL statements and operations and used to perform a discrete function, such as averaging.
Definition Mode	Mode in which user defines functions (programs) and stores them for future use.
Dimension	One of the independent axes of a shape. Dimensions are numbered from 1 to n for an n-dimensional object (origin 1) or 0 to n-1 (origin 0).
Dyadic Function	A function dealing with two arguments (left and right).
Element	A single scalar object located by a set of coordinates in an array.
Empty	A size-zero datum.
Expression (APL)	A niladic, monadic, or dyadic procedure; a value-producing expression; or a mixed output.
Face	The first or last plane "across" any dimension.
File	A workspace extension with components containing data objects.
Fill	Objects used to expand the size of a datum. Blanks (spaces) are used for character objects; zeroes (0's) are used for numeric objects.
Function	A value-producing operation on zero, one, or two arguments.

Term, Abbreviation, or Acronym	Definition
Identifier	A string starting with a letter, an underscored letter, or a delta (Δ) or underscored delta (⍙) and followed by any number of additional letters, deltas, or underscores.
Index	An integer specifying the position of a scalar element along a dimension of an array.
Index Position	The position of a plane relative to a dimension of datum. Index position "i" of a plane selects the i-th plane along a given dimension.
Index Sequence	The conventional order of the sets of indices of the elements of an array, whereby the last dimension steps through its values most rapidly. (As [1;1;1][1;1;2][1;1;3][1;2;1][1;2;2][1;2;3] etc.)
Iteration	A single execution of repetitive program steps or a loop.
Lamp Symbol ⍝	A prefix to denote comment (niladic text).
LSN	Logical Station Number (identifying number for each terminal station in network).
Library	Inactive workspaces stored for later use.
Matrix	A rank-2 datum (two dimensions).
MCS	Message Control System (data communications control system).
Monadic Function	A function dealing with only one argument (always right argument).
Niladic Function	A function dealing with no argument; produces no result.
n-Dimensional	A rank-n datum. (A datum having more than two dimensions.)

Appendix A

Glossary

Term, Abbreviation, or Acronym	Definition
Pendant Function	A function that is awaiting completion of another function that it called.
Plane	Any "slice" of a shaped object that is orthogonal to a given dimension of that object. A plane "across" the K-th dimension of an n-dimensional object is a (n-1)-dimensional object with all but the K-th dimension of the original retained. Thus a "plane" of a vector is a scalar element, and a "plane" of a matrix is a row or column.
Position	An index (location integer) to a list.
Prompt (system)	A response (from the APL/700 System) that is normally a five-space indentation on the terminal. The terminal is unlocked to accept user entries.
Rank	The number of dimensions of a datum. Scalars are rank 0, vectors are rank 1, matrices are rank 2, and n-dimensional arrays are rank n.
Scalar	A single data object without shape; that is, a rank-0 datum which may be either a number or character.
Shape	A vector specifying the number of planes along each dimension of a datum.
Shared Variable	A system variable that is shared among users and systems.
Size	The number of elements in an array.
Statement (APL)	An APL expression or comment.
String	A scalar or vector of characters.
Surrogate	A substitute, or secondary name as applied to shared-variable handling.
Suspended Function	A function which is currently being executed or whose execution was stopped for some reason other than a call to another function.

Appendix A

Glossary

Term, Abbreviations, or Acronym	Definition
System Variable	A variable used by APL/700 to specialize processing within a workspace (index origin, print position, comparison tolerance, and random link).
Text	Any string of characters.
Transaction	Cycle consisting of user entry, APL processing, and display of output, as required.
Type	Either character or numeric.
Vector	A rank-1 datum.
Vector Index	The index which selects each element in a vector or one index of the indices of an array which selects each element of a vector in an array which is described by varying the given index as the other indices remain constant.
Vector Origin	A sub-array selected by holding one vector index at the origin value.
Workspaces (active/stored)	Allocated blocks of disk and main memory storage for users to perform current transactions/operations and save data.

SUMMARY OF TRANSACTION, TYPING, AND ATTENTION CONVENTIONS

TRANSACTION CYCLE

1. SYSTEM INITIATES CYCLE BY TYPING OUT PROMPT AND UNLOCKING KEYBOARD.
2. USER SPECIFIES TRANSACTION BY TYPING IN TEXT.
3. SYSTEM COMPLETES TRANSACTION BY INTERPRETING ENTRY, TYPING OUT APPROPRIATE DATA OR ERROR MESSAGE, AND RETURNING TO STEP 1.

TYPING RULES

KEY	ACTION
CHARACTER	INSERT CHARACTER INTO TEXT AT POSITION OF CURSOR.
SPACE	POSITION CURSOR ONE SPACE TO RIGHT.
BACKSPACE	POSITION CURSOR ONE SPACE TO LEFT.
TAB	POSITION CURSOR RIGHTWARD TO NEXT TAB STOP.
LINEFEED	DISCARD TEXT ABOVE AND TO RIGHT OF CURSOR.
RETURN	TERMINATE USER ENTRY PORTION OF TRANSACTION.

TRANSACTION EDIT

1. SYSTEM EITHER (1) TYPES OUT TEXT, RETURNS, AND UNLOCKS KEYBOARD, OR (2) EXDENTS CURSOR, AND UNLOCKS KEYBOARD.
2. USER TYPES IN EDIT CONTROLS.
   INITIAL INPUT OF --
   - ATTENTION — SYSTEM ASSUMES MODIFICATION AT END OF TEXT, POSITIONS CURSOR TO COLUMN IMMEDIATLY TO RIGHT OF TEXT, UNLOCKS KEYBOARD AND PROCEEDS AT STEP 4.

   OTHERWISE INPUT OF --
   - '/' — UNDER CHARACTER OF TEXT 'DELETES'.
   - '.' — UNDER CHARACTER OF TEXT 'INSERTS'.
3. SYSTEM TYPES OUT REVISED TEXT, STOPPING BEFORE NEXT INSERTION POINT, AND UNLOCKS KEYBOARD.
4. USER ADDS TO, MODIFIES, OR COMPLETES CURRENT ENTRY BY USUAL TYPING RULES.
   INPUT OF --
   - ATTENTION — PROCEEDS AT STEP 3 IF CURSOR TO RIGHT OF CURRENT TEXT AND MORE ORIGINAL TEXT REMAINS.

ATTENTION CONVENTIONS

KEYBOARD STATE	ATTENTION INPUT IS	ACTION (SEE TRANSACTION EDIT FOR OVERRIDING USES)			
UNLOCKED	INITIAL	MODE	PROMPT	AFTER VALID ENTRY	AFTER ERRONEOUS ENTRY
		EXECUTION	FIVE SPACES	EDIT MOST RECENT APL EXPRESSION THIS LEVEL.	EDIT ERRONEOUS ENTRY.
		DEFINITION	[...]	EDIT MOST RECENT DEFINITION MODE ENTRY.	
		□	□: LF 3-SP	UNLOCK KEYBOARD.	
		⍞	USER DEFINED	UNLOCK KEYBOARD.	
	NON-INITIAL	SYSTEM LINEFEEDS, TYPES 'v', LINEFEEDS, AND UNLOCKS KEYBOARD. ACTION SAME AS LINEFEED FOR TYPING RULES.			
LOCKED	N.A.	SEQUENTIALLY INPUT ATTENTIONS MEAN:			
		DURING EXECUTION OF AN APL EXPRESSION: FIRST — SUSPEND EXECUTION AND ABORT QUEUED OUTPUT. SECOND — KILL ACTION.		OTHERWISE: ABORT QUEUED OUTPUT.	