

APL\360 User's Manual

```
3333300000000000\\\LLLLLPPPPP66666\\\AAAAAAAAAAAAAAAAAAAA    66666
3333300000000000\\\LLLLLPPPPP66666\\\AAAAAAAAAAAAAAAAAAAA    66666
3333300000000000\\\LLLLLPPPPP66666\\\AAAAAAAAAAAAAAAAAAAA    66666
PPPPP          AAAAA66666\\\66666\\\00000    66666LLLLL
PPPPP          AAAAA66666\\\66666\\\00000    66666LLLLL
PPPPP          AAAAA66666\\\66666\\\00000    66666LLLLL
      LLLLL66666    00000    AAAAA\\\        00000    \\\
      LLLLL66666    00000    AAAAA\\\        00000    \\\
      LLLLL66666    00000    AAAAA\\\        00000    \\\
AAAAA0000066666    66666    33333    6666600000    AAAAA
AAAAA0000066666    66666    33333    6666600000    AAAAA
AAAAA0000066666    66666    33333    6666600000    AAAAA
      P PPPP66666LLLLL00000\\\        P PPPP\\\        AAAAA00000
      P PPPP66666LLLLL00000\\\        P PPPP\\\        AAAAA00000
      P PPPP66666LLLLL00000\\\        P PPPP\\\        AAAAA00000
LLLLL33333PPPPPAAAAA\\\        P PPPP    6666600000\\\
LLLLL33333PPPPPAAAAA\\\        P PPPP    6666600000\\\
LLLLL33333PPPPPAAAAA\\\        P PPPP    6666600000\\\
33333      LLLLLLLLLL\\\AAAAALLLLL    AAAAA\\\
33333      LLLLLLLLLL\\\AAAAALLLLL    AAAAA\\\
33333      LLLLLLLLLL\\\AAAAALLLLL    AAAAA\\\
      AAAAA\\\3333366666AAAAALLLLL33333PPPPP    66666    00000
      AAAAA\\\3333366666AAAAALLLLL33333PPPPP    66666    00000
      AAAAA\\\3333366666AAAAALLLLL33333PPPPP    66666    00000
66666\\\66666\\\PPPPPLLLLLPPPPP\\\00000    AAAAA
66666\\\66666\\\PPPPPLLLLLPPPPP\\\00000    AAAAA
66666\\\66666\\\PPPPPLLLLLPPPPP\\\00000    AAAAA
AAAAALLLLLPPPPP33333LLLLL\\\66666LLLLLAAAAAPPPP\\\AAAA
AAAAALLLLLPPPPP33333LLLLL\\\66666LLLLLAAAAAPPPP\\\AAAA
AAAAALLLLLPPPPP33333LLLLL\\\66666LLLLLAAAAAPPPP\\\AAAA
      666666666666666600000333330000033333LLLLL66666\\\PPPPP33333
      666666666666666600000333330000033333LLLLL66666\\\PPPPP33333
      666666666666666600000333330000033333LLLLL66666\\\PPPPP33333
66666          00000\\\        33333LLLLL000003333300000AAAAA
66666          00000\\\        33333LLLLL000003333300000AAAAA
66666          00000\\\        33333LLLLL000003333300000AAAAA
AAAAA          AAAAAAAAAA00000LLLLL          \\\LLLLL    P PPPP66666
AAAAA          AAAAAAAAAA00000LLLLL          \\\LLLLL    P PPPP66666
AAAAA          AAAAAAAAAA00000LLLLL          \\\LLLLL    P PPPP66666
```

The cover design uses a random pattern generated by the following APL function:

```
∇ Z+M G C
[1] Z+((2+M)×2+M)ρ 2 4 1 3 ⍋((M[3 4],ρC)ρC)[; ;?M[1 2]ρρC]
∇
```

This function was suggested by an algorithm due to Roger Frey (IBM Research and Yale University). The particular usage was: 13 13 3 5 G ' APL\360 '

The body of the manual was edited and composed on APL\360 using variations of a text-processing package designed and implemented by M. M. Zryl (IBM Research), and patterned after a Fortran system due to A. P. Mullery (IBM Research).

APL\360: User's Manual

A. D. Falkoff

K. E. Iverson

© International Business Machines Corporation, 1968

ACKNOWLEDGEMENTS

The APL language was first defined by K. E. Iverson in A Programming Language (Wiley, 1962) and has since been developed in collaboration with A. D. Falkoff. The APL\360 Terminal System was designed with the additional collaboration of L. M. Breed, who, with R. D. Moore*, also designed the S/360 implementation. The system was programmed for S/360 by Breed, Moore, and R. H. Lathwell, with continuing assistance from L. J. Woodrum†, and contributions by C. H. Brenner, H. A. Driscoll‡, and S. E. Krueger‡. The present implementation also benefitted from experience with an earlier version, designed and programmed for the IBM 7090 by Breed and P. S. Abrams**.

The development of the system has also profited from ideas contributed by many other users and colleagues, notably E. E. McDonnell, who suggested the notation for the signum and the circular functions.

In the preparation of the present manual, the authors are indebted to L. M. Breed for many discussions and suggestions; to R. H. Lathwell, E. E. McDonnell, and J. G. Arnold †† for critical reading of successive drafts; and to Mrs. G. K. Sedlmayer and Miss Valerie Gilbert for superior clerical assistance.

A special acknowledgement is due to John L. Lawrence, who provided important support and encouragement during the early development of APL implementation, and who pioneered the application of APL in computer-related instruction.

* I. P. Sharp Associates, Toronto, Canada.

† General Systems Architecture, IBM Corporation, Poughkeepsie, N.Y.

‡ Science Research Associates, Chicago, Illinois.

** Computer Science Department, Stanford University, Stanford, California.

†† Industry Development, IBM Corporation, White Plains, N.Y.

TABLE OF CONTENTS

PART 1: GAINING ACCESS

PHYSICAL EQUIPMENT	1.1
Preferred features	
THE APL CHARACTER SET	1.3
Use of other character sets	
THE RECORDING TERMINAL	1.4
ESTABLISHING A CONNECTION	1.4
Set up terminal, Dial computer	
ENTRIES FROM THE KEYBOARD	1.6
Transmission signals, Mistakes, Transmission errors, Special features of IBM 1050 terminals	
STARTING AND ENDING A WORK SESSION	1.8
Sign-on, Limited use of the system, Disconnect dial-up connection, Break any connection	

PART 2: SYSTEM COMMANDS

WORKSPACES AND LIBRARIES	2.1
Workspaces, Libraries	
NAMES	2.2
Local and global significance	
LOCKS AND KEYS	2.3
ATTENTION	2.4
USE OF SYSTEM COMMANDS	2.5
Classification of commands, Normal response and trouble reports, Clear workspace, Summary	
TERMINAL CONTROL COMMANDS	2.8
Forced endings, The CONTINUE workspace, Interrupted activities, Detailed description	
WORKSPACE CONTROL COMMANDS	2.13
Application Packages, Groups, Detailed description	
LIBRARY CONTROL COMMANDS	2.22
Continuity of work, Workspace identification, Library and account numbers, Storage allotment, Use of the CONTINUE workspace, Purging a workspace, Detailed description	
INQUIRY COMMANDS	2.28
User codes, Detailed description	
COMMUNICATION COMMANDS	2.32
Detailed description	

PART 3: THE LANGUAGE

FUNDAMENTALS	3.1
Statements, Scalar and vector constants, Names and spaces, Overstriking and erasure, End of statement, Order of execution, Error reports, Names of primitive functions	
SCALAR FUNCTIONS	3.5
Monadic and dyadic functions, Vectors, Index generator	
DEFINED FUNCTIONS	3.10
Introduction, Branching, Local and global variables, Explicit argument, Explicit result, The forms of defined functions, Use of defined functions, Recursive function definition, Trace control	
MECHANICS OF FUNCTION DEFINITION	3.17
Labels, Revision, Display, Line editing, Reopening function definition, Locked functions, Deletion of functions and variables, System command entered during function definition	
SUSPENDED FUNCTION EXECUTION	3.20
Suspension, State indicator, Stop control	
HOMONYMS	3.22
Variable names, Function names	
INPUT AND OUTPUT	3.23
Evaluated input, Character input, Escape from input loop, Normal output, Heterogeneous output	
RECTANGULAR ARRAYS	3.26
Introduction, Vectors dimension catenation, Matrices dimension ravel, Reshape, Uses of empty arrays, Indexing, Indexing on the left, Index origin, Array output	
FUNCTIONS ON ARRAYS	3.33
Scalar functions, Reduction, Inner product, Outer product	
MIXED FUNCTIONS	3.37
Introduction, Transpose, Monadic transpose, Rotate, Reverse, Compress, Expand, Decode, Encode, Index of, Membership, Take and drop, Grade up and down, Deal, Comments	
MULTIPLE SPECIFICATION	3.45
SYSTEM DEPENDENT FUNCTIONS	3.46

PART 4: LIBRARY FUNCTIONS

APPENDIX A

SAMPLE TERMINAL SESSION

APPENDIX B

ADVANCED EXAMPLES

BIBLIOGRAPHY

LIST OF ILLUSTRATIONS

	<u>Page</u>
Table 1.1 RECOMMENDED FEATURES AND OPTIONS FOR TERMINALS	1.2
Figure 1.2 APL\360 KEYBOARD	1.3
Table 1.3 TELEPHONE NUMBERS	1.5
Table 2.1 SYSTEM COMMANDS	2.7
Table 3.1 ERROR REPORTS	3.4
Table 3.2 PRIMITIVE SCALAR FUNCTIONS	3.6-7
Table 3.3 FORMS OF DEFINED FUNCTIONS	3.15
Table 3.4 DIMENSION AND RANK VECTORS	3.27
Table 3.5 IDENTITY ELEMENTS OF PRIMITIVE SCALAR DYADIC FUNCTIONS	3.34
Table 3.6 INNER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTIONS f AND g	3.36
Table 3.7 OUTER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTION g	3.36
Table 3.8 PRIMITIVE MIXED FUNCTIONS	3.38
Table 3.9 TRANSPOSITION	3.40
Table 3.10 SYSTEM DEPENDENT FUNCTIONS	3.47

PART 1

GAINING ACCESS

An APL\360 System comprises a central computer and an indefinite number of typewriter-like terminals. A certain number of these terminals may be simultaneously linked to the computer, according to the number and type of access ports on the computer.

This part of the manual describes the terminal equipment required for interacting with the system, tells how to establish a connection between a terminal and the central computer, and gives, in simplest form, the procedures for starting and ending a work session.

PHYSICAL EQUIPMENT

A remote terminal for use with the system must be either an IBM 2741 Communications Terminal, an IBM 2740-1 Communications Terminal equipped with the Transmit Control feature, or an IBM 1050 Teleprocessing Terminal. It may connect to the central computer through the dial-up telephone network, by a leased telephone line, or by private wire.

Dial-up connections are effected by means of a Western Electric Dataset #103A-2 or the equivalent, or a compatible acoustic coupler. A leased telephone line connection requires the use of a Western Electric Dataset #103F-2 or the equivalent. A direct-wired connection is effected by means of an appropriate IBM line adapter (modem). In the last case, two-wire connections should be avoided, if possible, since their use rules out an interrupt facility.

Preferred features. The APL\360 system will work with many variations of the terminal types given above, but certain features and options are desirable. Dial-up connections provide the greatest flexibility, both in overall system configuration, and in certain details of operation. Similarly, although the APL printing element is based on a 12-pitch font, and is available in both Selectric® and PTTC/BCD keyboard encoding (i.e., the correspondence between

FEATURE OR OPTION	1050	2740-1	2741
Control Unit	1051-2		
Voltage (115 AC), Non-lock plug	9881	9881	9881
Dataset Attachment	9114	9114	9114
Dial Up	NR	3255	3255
Transmit Control	NR	8028	NR
Automatic EOB	RPQ E27283	Do not use	NR
Typamatic Keys	NA	NA	8341
Interrupt	RPQ E27428	RPQ F17913	4708
Text Time-out Suppression	9698	NR	NR
First Printer Attachment	4408	NR	NR
Automatic Ribbon Shift Select	1295	NA	NA
Typing Table	9705	NR	NR
Printer-Keyboard	1052-2		
APL Printing Element, PTTC/BCD or Standard Selectric®	1167988 NA	1167988 1167987	1167988 1167987
Keys, APL Keyboard	RPQ M40174	RPQ M40174	RPQ M40174
Character Spacing, 10 per inch	9104	9104	9104
Line Feeding, 6 per inch	9435	9435	9435
Accelerated Carrier Return	1006	NA	NA
Notes. NR: feature is standard equipment, or is not required. NA: not available (July 1968). The numbers are IBM-domestic identifications.			

Table 1.1: RECOMMENDED FEATURES AND OPTIONS FOR TERMINALS

keyboard layout and character positions on the printing element), specification of 10-pitch character spacing and Selectric® encoding will allow a greater variety of printing elements to be used with the terminal. While it is not essential, the convenience of having the interrupt feature cannot be overestimated.

Paper tape equipment (1054-1 Reader and 1055-1 Punch) and punched-card equipment (1056-1 Reader and 1057-1 Punch) can be used with IBM 1050 terminals. The punched-card facilities should have Extended Character features 3861 and 3860, for reader and punch, respectively.

IBM identifications for recommended terminal features and options are given in Table 1.1. Complete specifications for terminals, and information on other options, should be obtained from local IBM representatives.



Figure 1.2: APL\360 KEYBOARD

THE APL CHARACTER SET

The APL\360 keyboard is shown in Figure 1.2. The numerals, alphabetic characters, and punctuation marks appear in their usual places, although the alphabet is used in only a single case: letters print as upper-case italics, but are produced only when the keyboard is in lower-case position (i.e., not shifted).

The special characters, most of which are produced with the keyboard shifted, generally have some mnemonic connection with their alphabetic or numeric correspondents. This may be appearance (ω over W), Greek-Roman equivalence (ρ over R), sequence ($< \leq = \geq > \neq$ over 3 4 5 6 7 8), or some -- possibly far fetched -- relationship between the APL function represented by the symbol and the letter ($*$ over P for power, ' over K for "kwote", and \lceil over S for ceiling).

Use of other character sets. The part numbers of APL printing elements are given in Table 1.1. However, any printing element may be used with the APL system, since the encoded characters generated by the keyboard and transmitted to the computer are independent of the particular element mounted on the terminal. Subject to programmed intervention, the transmitted information will always be interpreted according to the APL keyboard characters.

Non-APL printing elements are frequently useful in conjunction with special-purpose APL programs designed to exploit their character sets. Also, any element that matches the keyboard encoding (Selectric® or PTTC/BCD) of the terminal can be used for straightforward numerical work, since letters and digits print properly with such elements. The visual interpretation of complex APL expressions is, of course, awkward with any but an APL printing element.

THE RECORDING TERMINAL

As connections with remote terminals are established and broken, and users start and end work sessions, a printed record of this traffic is generated at the system's recording terminal. This terminal, which is usually, but not necessarily, located at the central computer site, is ordinarily attended by an APL Operator who monitors the operation of the system, and provides a common point of contact for users.

There are certain supervisory functions, essential to the operation of APL\360, which can be effected only from the recording terminal. Thus, this terminal holds a privileged position relative to others. The enrollment of new users, and the allocation of library space, are examples of this kind of function.

ESTABLISHING A CONNECTION

The directions that follow assume the use of a dial-up connection with a dataset. Instructions for the use of acoustic couplers should be obtained from their suppliers. Where terminals are connected to the computer by leased lines or private wires, instructions on dialing procedure (EC2) are irrelevant, but local sources of information should be consulted for equivalent procedures.

ACTION

NOTES

EC1. Set up terminal:

Insert paper, mount an APL printing element, connect terminal to power source, and set switches as follows:

IBM 2741 or 2740 Terminal

LCL/COM
Power

COM
ON

The power switch is at the right of the keyboard. On 2741's, the LCL/COM switch is on the left side of the terminal stand, toward the rear; on 2740's, it is to the right of the power switch.

IBM 1050 Terminal

ATTEND/UNATTEND	ATTEND
Keyboard	SEND
Printer	SEND-RECEIVE
Reader 1	OFF
Punch 1	OFF
EOB	MANUAL
Line test	OFF
Single step	OFF
Line control	ON
Power	ON

Not all 1050's have all switches; those present must be set as indicated. The states of switches not listed here are immaterial.

If it is known that RPO E27283 (see Table 1.1) is installed, set the EOB switch to AUTO.

The line control switch is inside the rear door of the 1051 Control Unit. The power switch is on the left side of the control unit, toward the front.

On 2741 and 2740 terminals, test to see if the keyboard is locked by trying the shift key. If the key is operable, press the carrier return and test again.

If the keyboard does not lock after a carrier return, check the switches and try once more. If the switches are set properly and the keyboard remains unlocked, the terminal is faulty.

EC2. Dial computer:

Set the telephone pushbutton switch to TALK and follow ordinary dialing procedure. After two rings, at most, the telephone will respond with a steady, high-pitched tone.

Telephone numbers are given in Table 1.3. If the line is busy, try a different number or call the APL Operator to inquire about an open line.

123 456-7890	123 456-7890
Insert a table of access telephone numbers here.	
An assistance number should be included.	
APL Operator:	123 456-7890

Table 1.3: TELEPHONE NUMBERS

Promptly set the pushbutton switch to DATA by holding the DATA button down firmly for a moment and then releasing.

Cradle the handset.

The DATA button should light, and will remain lit as long as the terminal is connected to the computer. If it does not light, check the power connection to the dataset. If it lights, but quickly goes out, check the power connection to the terminal, the cable connection to the dataset, and the switch settings on the terminal. Then retry from EC1.

Response: The keyboard will unlock, indicating that the computer is ready to accept an entry from the terminal.

The connection established by the foregoing procedure is only tentative, and will be broken by the central computer if further communication does not take place within 60 seconds. Therefore, the next step -- the sign-on procedure (EC3) given below -- should be executed promptly.

ENTRIES FROM THE KEYBOARD

After a connection is established, normal communication between a terminal and the central computer is carried on by means of entries from the typewriter keyboard, which alternately locks and unlocks as each entry is made and the computer completes its work. The general procedure is to type an instruction or command, strike the carrier return to indicate the end of the message, and follow this by a transmission signal.

Transmission signals. The transmission signal is generated differently, according to the terminal type and its equipment:

2741. A transmission signal is automatically generated in the proper sequence (i.e., after the carrier return signal) when the RETURN key is struck.

2740. The transmission signal is produced by striking the EOT key after the RETURN key. (Do not use the the EOB key, or the automatic EOB feature available on these terminals.)

1050. On terminals equipped with an automatic EOB RPQ (see Table 1.1), the transmission signal is produced automatically when the RETURN key is struck. Otherwise, an EOB must be produced manually, by striking the numeral-5 key, while the key marked ALTN CODING is held down. (Note that the automatic EOB feature available for 1050 terminals cannot be used with APL\360.)

A transmission signal does not cause a character to be printed, and its omission will therefore be evidenced only by the state of the terminal: the keyboard will remain unlocked, and no response will be forthcoming from the system.

In the remainder of this manual the need for carrier return and transmission signal will not be explicitly mentioned, since they are required for every entry.

Mistakes. Before the carrier return (and transmission signal) that completes an entry, errors in typing can be corrected as follows: backspace to the point of error and then depress the linefeed button (marked ATTN on 2741 terminals). This will have the effect of erasing everything to the right of, and including, the position of the carrier. The corrected text can be continued from that point, on the new line.

This procedure can be used at any time once the sign-on (EC3) has been accomplished. In case of error in the sign-on itself, the entry should be made as is. The system will provide an appropriate trouble report, following which, a correct entry may be made.

Transmission errors. There are occasional transient failures in the communication between a terminal and the central computer. If the failure occurs during the transmission from the terminal, the system will respond with a resend signal: on 1050 terminals, the RESEND warning light will go on, and on other terminals the message *RESEND* will be printed. In any case, the last entry from the keyboard must be repeated. The warning light on the 1050 should first be extinguished by pressing the adjacent button.

Failures in the other direction are usually evidenced by the appearance of a spurious character, whose presence in the printed output is obvious in most contexts. However, there is no absolutely certain way of detecting such a failure.

Special features of IBM 1050 terminals. The keyboard of a terminal equipped with a REQUEST button will not unlock, when it otherwise should, until the button is depressed. On terminals equipped with a timer, the keyboard will lock before an entry is completed if approximately 18 seconds have elapsed since the last keyboard action. Locking can be forestalled by occasionally striking the shift key, but if it does happen, the keyboard can be forced to unlock by flipping the line-control switch inside the 1051 Control Unit to OFF, and back to ON.

If a terminal is to be used exclusively with APL\360, the Keyboard Request feature should be removed, and the Text Time-out Suppression feature should be added.

STARTING AND ENDING A WORK SESSION

Each user of the system is assigned an account number. This number is used to effect the sign-on that initiates a work session; serves to partially identify any work that the user may store in the system between sessions; and is used for accounting or billing purposes.

If the account number is not known, or if one of the trouble reports given below is encountered and not understood, a message of inquiry can be sent to the APL Operator. This is accomplished by entering)OPR followed by a space and one line (not exceeding 120 characters) of an appropriate text.

Such a message can be sent at any time after a connection has been established. It causes the keyboard to lock, awaiting a reply. If no reply is forthcoming, (and the sign-on has not been completed), the connection will have to be broken and re-established before further communication with the system is possible. (After the sign-on, the keyboard may be unlocked by an attention signal, described in Part 2.)

<u>ACTION</u>	<u>NOTES</u>
<u>EC3. Sign on:</u> Enter) followed by an account number, with a key (i.e., a colon and password), if required.	The use of passwords as locks and keys is described in Part 2. A new user will have been advised if a key is required for his first sign-on.

Effect:

1. A workspace will be activated for the terminal.
2. Accumulation of time charges will begin.

Response:

1. A broadcast message from the APL Operator may be printed.
2. The port number, time of day, date, and user name associated with the account number will be printed on one line. The system identification will be printed on another line.

A workspace can be thought of as both a notebook and a scratch pad. The details are explained in Part 2.

Trouble reports:

NUMBER NOT IN SYSTEM

means either exactly what it says, or that the number has a lock associated with it and the wrong key was used. The APL Operator should be consulted if help is required.

INCORRECT SIGN-ON

means the form of the transmitted command was faulty. Retry with a properly formulated sign-on.

ALREADY SIGNED ON

means that a work session is already in progress at the terminal. To start a session with a different account number, use command TC5 (see Part 2), which ends the current session but holds the connection, and retry from the beginning of EC3.

NUMBER IN USE

means just that, or a temporary condition due to delays in the central computer. Retry from EC2 after two minutes. If the condition persists, notify the APL Operator.

NUMBER LOCKED OUT

means that authorization for use of that number has been withdrawn.

3. *SAVED*, followed by the time of day and date that the activated workspace was last stored.

This response will be omitted if the activated workspace is clear (i.e., not holding information). If the response is given, the workspace is named *CONTINUE*. The use of workspace names is explained in Part 2.

4. The keyboard will be unlocked.

If this is the only response, a transmission error has occurred, or the entry did not start with an APL right parenthesis. In either case, the entry should be repeated in correct form. If the condition persists, retry from EC2, possibly dialing a different number.

A work session is started, and the full APL system becomes available, once the sign-on is accomplished. Any system command of Part 2 or APL operation of Part 3 may now be entered for execution.

Limited Use of the System. No system command other than the sign-on given here is required in order to make use of Part 3, and the reading of Part 2 may therefore be deferred if only casual or restricted use is to be made of the system. For the purposes of such use, a work session may conveniently be terminated by one of the following procedures:

ACTION

NOTES

EC4. Disconnect dial-up connection:
Set the power switch to OFF.

Use this procedure for dial-up connections only.

Effect:

1. The active workspace will be stored under the name *CONTINUE*.

If the workspace is clear, it will not be stored at this time. If it is stored, it will be automatically re-activated when the same account number is next used to sign on. See note for EC3, Response 3.

2. The duration of the work session and the amount of computer time used will be noted internally for later accounting.

3. The connection to the central computer will be broken.

Response: None.

The DATA light will go out.

EC5. Break any connection:
Enter)*CONTINUE*

This is command TC4,
detailed in Part 2.

Effect:

1. 2. and 3. The same as for EC4.

Response:

1. Time of day and date,
followed by *CONTINUE*

Trouble reports:

*NOT WITH OPEN DEFINITION
INCORRECT COMMAND*

The meanings of these reports, and corrective actions for them, are given in Part 2.

2. The port number, time of day, date, and user code will be printed.

User codes comprise three characters which partially identify users. Their use is explained in Part 2.

3. Accounting information will be printed.

If a dial-up connection is being used, the DATA light will go out.

PART 2

SYSTEM COMMANDS

APL operations deal with transformations of abstract objects, such as numbers and symbols, whose practical significance, as is usual in mathematics, depends upon the (arbitrary) interpretation placed upon them. System commands in the APL\360 System, on the other hand, have as their subject the structures which comprise the system, and control functions and information relating to the state of the system, and therefore have an immediate practical significance independent of any interpretation by the user.

In this Part the structure of the APL\360 system is described, and various notions essential to the understanding of system commands are introduced. Finally, the complete set of system commands is described in detail.

WORKSPACES AND LIBRARIES

Workspaces. The common organizational unit in the APL\360 system is the workspace. When in use, a workspace is said to be active, and it occupies a block of working storage in the central computer. The size of the block, which is preset at a fixed value for a given system, determines the combined working and storage capacity of each workspace in that system. Part of each workspace is set aside to serve the internal workings of the system, and the remainder is used, as required, for storing items of information and for containing transient information generated in the course of a computation.

An active workspace is always associated with a terminal during a work session, and all transactions with the system are mediated by it. In particular, the names of variables (data items) and defined functions (programs) used in calculations always refer to objects known by those names in the active workspace; information on the progress of program execution is maintained in the state indicator of the active workspace; and control information affecting the form of output is held within the active workspace.

Libraries. Inactive workspaces are stored in libraries, where they are identified by arbitrary names. They occupy space in secondary storage facilities of the central computer and cannot be worked with directly. When required, copies of stored workspaces can be made active, or selected information may be copied from them into an active workspace.

Libraries in APL\360 are either private or public. Private libraries are associated with individual users of the system, and are identified by the user's account number. Access to them by other users is restricted in that one user may not store workspaces in another person's library, nor can he obtain a listing of the workspaces already stored there. However, one user may activate a copy of another user's (unlocked) workspace if he knows the library number and workspace name.

Public libraries are identified by numbers below 1000. They are not associated with individual users, although certain ones may be reserved by general agreement for groups of people working cooperatively. Anyone may store workspaces in a public library, and a listing of workspace names is available upon request if the library number is known. However, a workspace stored in a public library remains under the control of the user who put it there, and cannot be altered by others.

NAMES

Names of workspaces, functions, variables, and groups (see workspace control commands) may be formed of any sequence of alphabetic (A to Z, and A to Z) and numeric (0 to 9) characters that starts with an alphabetic and contains no blank. Only the first 11 characters of workspace names, and the first 77 characters of other names are significant. Longer names may be used, but additional characters beyond these limits are ignored.

The environment in which APL operations take place is bounded by the active workspace. Hence, the same name may be used to designate different objects (i.e., groups, functions, or variables) in different workspaces, without interference. Also, since workspaces themselves are never the subject of APL operations, but only of system commands, it is possible for a workspace to have the same name as an object it holds. However, the objects within a workspace must have distinct names, except as explained below.

Local and global significance. In the execution of defined functions it is often necessary to work with intermediate results which have no significance either before or after the function is used. To avoid cluttering the workspace with a multitude of variables introduced for such transient purposes, and to allow greater freedom in the choice of names, the function definition process (see Part 3) provides a facility for designating certain variables as local to the function being defined. Variables not so designated, and all functions and groups, are said to be global.

A local variable may have the same name as a global object, and any number of variables local to different functions may have the same name.

During the execution of a defined function, a local variable will supersede a function or global variable of the same name, temporarily excluding it from use. If the execution of a function is interrupted (leaving it either suspended, or pendent, see Part 3), the local variables retain their dominant position, during the execution of subsequent APL operations, until such time as the halted function is completed. System commands, however, continue to reference the global homonyms of local variables under these circumstances.

LOCKS AND KEYS

Stored workspaces and the information they hold can be protected against unauthorized use by associating a lock, comprising a colon and a password of the user's choice, with the name of the workspace, when the workspace is stored. In order to activate a locked workspace or copy any information it contains, a colon and the password must again be used, as a key, in conjunction with the workspace name. Listings of workspace names, including those in public libraries, never give the keys, and do not overtly indicate the existence of a lock.

Account numbers can be similarly protected by locks and keys, thus maintaining the security of a user's private library and avoiding unauthorized charges against his account.

Passwords for locks and keys may be formed of any sequence of alphabetic and numeric characters up to eight characters long, without blanks. Characters beyond the eighth are ignored. In use as either a lock or key, a password follows the number or name it is protecting, from which it is set off by a colon.

ATTENTION

Printed output at a terminal can be cut off, or the execution of an APL operation can be interrupted, and control returned to the user, by means of an attention signal. Since the keyboard is locked during printing or computing, the signal must be generated by means other than one of the standard keys.

On terminals equipped with an interrupt feature, the attention signal is generated by depressing the appropriate key once, firmly. On IBM 2741 terminals this key is usually of a distinctive color, and is marked ATTN. (The same key is used for linefeed when the keyboard is not locked.)

For terminals not so equipped, the attention signal is generated by momentarily interrupting the connection to the central computer. The method depends upon the type of connection:

with dial-up telephones, uncradle the handset, set the pushbutton switch to TALK for two to three seconds, and then reset it to DATA;

with leased telephone lines, set the terminal power switch to OFF and then back to ON, with deliberate speed.

If the connection is broken, in either case, for more than five seconds, the central computer will interpret this as a signal to end the work session and will execute action EC4 of Part 1.

Following an attention signal the keyboard will unlock, and the type carrier will return to the normal position for input (six spaces from the left margin). If the carrier does not do this, enter blank lines repeatedly until it does. In some cases a line will be printed before the keyboard unlocks, telling where a function in progress was interrupted.

Except for communication commands (and then only if the delivery of a message is delayed), the execution of system commands, once entered, cannot be interrupted. However, the printed responses or trouble reports following a system command can be suppressed by a properly timed attention signal.

USE OF SYSTEM COMMANDS

System commands and APL operations are distinguished functionally by the fact that system commands can be called for only by individual entries from the keyboard, and cannot be executed dynamically as part of a defined function. They are distinguished in form by the requirement that system commands be prefixed by a right parenthesis, which is a syntactically invalid construction in APL.

There is some system control which it may be desirable to exert dynamically, and there are some items of system information which can be profitably used during the execution of a program. For these purposes APL\360 provides appropriate system-dependent functions and library functions, which can be used like other APL operations. These functions are described in Part 3 and Part 4, respectively. Where a system command duplicates the action of one of them, this fact will be noted in the description of the system command in this Part.

All system commands can be executed when the terminal is in the execution mode, in which APL operations are executed forthwith upon entry. However, in definition mode, in which sequences of operations are being composed into functions for later execution, commands which call for storing a copy of the workspace, or which might otherwise interfere with the definition process itself, are forbidden. (The two terminal modes are treated more fully in Part 3.)

Classification of commands. System commands are conveniently grouped into five classes with regard to their effect upon the state of the system:

1. Terminal control commands affect the relation of a terminal to the system.
2. Workspace control commands affect the state of the active workspace.
3. Library control commands affect the state of the libraries.
4. Inquiry commands provide information without affecting the state of the system.
5. Communication commands effect the transmission of messages among terminals.

The text that follows is based upon this classification, although it will be seen that certain of the terminal control commands also affect the libraries, and one of the library control commands may sometimes affect the state of the active workspace.

Normal responses and trouble reports. Any entry starting with a right parenthesis will be interpreted by the system as an attempt to execute a system command. When the command is successfully executed, the normal response, if any, will be printed. This response is given in the description of the action for each command.

If, for any reason, a command cannot be executed, an appropriate trouble report will be printed. The most common report is *INCORRECT COMMAND*. This means that the command was incomplete, mis-spelled, used a wrong modifier, or was otherwise malformed. The corrective action in every case is to enter a properly composed command. The meanings and corrective actions for other trouble reports are given in the notes accompanying the description of each command.

Clear workspace. There are certain transient failures of the system which cause the active workspace to be destroyed. If this should occur, the message *CLEAR WS* will be printed, indicating that the active workspace has been replaced by a clear workspace. (The attributes of a clear workspace are given in the section on workspace control commands, see WC1.) This situation rarely arises, but the probability of its occurrence is slightly higher during the execution of system commands.

Summary. The purposes, forms, responses, and trouble reports for all system commands are summarized in Table 2.1. Where the first word of a command form is more than four characters long, only the first four are significant. The others are included only for mnemonic reasons, and may be dropped or replaced, as desired. For example, *)CLEAR*, *)CLEA*, *)CLEAVER*, etc., are all equivalent.

In general, the elements of a command form must be separated by one (or more) spaces. Spaces are not required immediately following the right parenthesis, or on either side of the colon used with passwords, but can be used without harm.

Reference and Purpose COMMAND FORM ^{1,2,3}	NORMAL RESPONSE	TROUBLE REPORTS ⁴
TC1 Sign on designated user and start a work session.)NUMBER [KEY]	[TEXT]; PORT,TIME,DATE,USER; SYSTEM; [SAVED,TIME,DATE]	1 2 3 4 5
TC2 End work session.)OFF [LOCK]	PORT,TIME,DATE,USER CODE; TIME USED	16
TC3 End work session and hold dial-up connection.)OFF HOLD [LOCK]	PORT,TIME,DATE,USER CODE; TIME USED	16
TC4 End work session and store active workspace.)CONTINUE [LOCK]	[TIME,DATE,CONTINUE]; PORT,TIME,DATE,USER CODE; TIME USED	6 16
TC5 End work session, store active workspace, and hold dial-up connection.)CONTINUE HOLD [LOCK]	[TIME,DATE,CONTINUE]; PORT,TIME,DATE,USER CODE; TIME USED	6 16
WC1 Activate a clear workspace.)CLEAR	CLEAR WS	16
WC2 Activate a copy of a stored workspace.)LOAD WSID [KEY]	SAVED,TIME,DATE	7 8 16
WC3 Copy a global object from a stored workspace.)COPY WSID [KEY] NAME	SAVED,TIME,DATE	6 7 8 9 10 16
WC3a Copy all global objects from a stored workspace.)COPY WSID [KEY]	SAVED,TIME,DATE	6 7 8 10 16
WC4 Copy a global object from a stored workspace, protecting active workspace.)PCOPY WSID [KEY] NAME	SAVED,TIME,DATE; [NOT COPIED:,LIST OF OBJECTS]	6 7 8 9 10 16
WC4a Copy all global objects from a stored workspace, protecting active workspace.)PCOPY WSID [KEY]	SAVED,TIME,DATE; [NOT COPIED:,LIST OF OBJECTS]	6 7 8 10 16
WC5 Gather objects into a group.)GROUP NAME[S]	NONE	11 16
WC6 Erase global objects.)ERASE NAME[S]	[NOT ERASED:,LIST OF OBJECTS]	16
WC7 Set index origin for array operations.)ORIGIN INTEGER,0-1	WAS,FORMER ORIGIN	16
WC8 Set maximum for significant digits in output.)DIGITS INTEGER,1-16	WAS,FORMER MAXIMUM	16
WC9 Set maximum width for an output line.)WIDTH INTEGER,30-130	WAS,FORMER WIDTH	16
WC10 Change workspace identification.)WSID WSID	WAS,FORMER WSID	16
LC1 Re-store a copy of the active workspace.)SAVE	TIME,DATE,WSID	6 12 13 14 16
LC1a Store a copy of the active workspace.)SAVE WSID [LOCK]	TIME,DATE	6 12 13 14 16
LC2 Erase a stored workspace.)DROP WSID	TIME,DATE	7 14 16
IQ1 List names of defined functions.)FNS [LETTER]	FUNCTION NAMES	16
IQ2 List names of global variables.)VARS [LETTER]	VARIABLE NAMES	16
IQ3 List names of groups.)GRPS [LETTER]	GROUP NAMES	16
IQ4 List membership of designated group.)GRP NAME	FUNCTION NAMES,VARIABLE NAMES	16
IQ5 List halted functions (state indicator).)SI	SEQUENCE OF HALTED FUNCTIONS	16
IQ6 List halted functions and associated local variables (augmented state indicator).)SIV	SEQUENCE OF HALTED FUNCTIONS WITH NAMES OF LOCAL VARIABLES	16
IQ7 Give identification of active workspace.)WSID	WSID	16
IQ8 List names of workspaces in designated library.)LIB [NUMBER]	NAMES OF STORED WORKSPACES	14 16
IQ9 List ports in use and codes of connected users.)PORTS	PORT NUMBERS AND ASSOCIATED USER CODES	16
IQ10 List port numbers associated with designated user code.)PORTS CODE	PORT NUMBERS	16
CM1 Address text to designated port.)MSGN PORT [TEXT]	SENT	15 16
CM2 Address text to designated port, and lock keyboard.)MSG PORT [TEXT]	SENT	15 16
CM3 Address text to recording terminal (APL Operator).)OPRN [TEXT]	SENT	15 16
CM4 Address text to recording terminal (APL Operator), and lock keyboard.)OPR [TEXT]	SENT	15 16
Notes: 1. Items in brackets are optional. 2. KEY or LOCK: a password set off from preceding text by a colon. 3. WSID: library number and workspace name, or workspace name alone, as required. 4. See insert table of trouble report forms.		

TROUBLE REPORT FORMS	
1	NUMBER NOT IN SYSTEM
2	INCORRECT SIGN-ON
3	ALREADY SIGNED ON
4	NUMBER IN USE
5	NUMBER LOCKED OUT
6	NOT WITH OPEN DEFINITION
7	WS NOT FOUND
8	WS LOCKED
9	OBJECT NOT FOUND
10	WS FULL
11	NOT GROUPED, NAME IN USE
12	NOT SAVED, WS QUOTA USED UP
13	NOT SAVED, THIS WS IS WSID
14	IMPROPER LIBRARY REFERENCE
15	MESSAGE LOST
16	INCORRECT COMMAND

Table 2.1: SYSTEM COMMANDS

TERMINAL CONTROL COMMANDS

There is one command for starting a work session, and there are four commands for ending one. The variations in ending allow for automatically storing a copy of the active workspace, and for holding a dial-up telephone connection to the central computer for an immediate start of another work session. The starting command has been described in Part 1 (EC3).

Forced endings. Any action that interrupts a telephone connection for more than five seconds will cause the work session to end, and usually cause a copy of the active workspace to be stored. This provides a safeguard against loss of work in case of failure in the telephone circuits, or accidental loss of power at the terminal. It is also the basis of the disconnect action described in EC4 of Part 1.

A work session can also be stopped remotely, from the system's recording terminal, in an action known as a bounce. As in a disconnect, a copy of the active workspace is usually stored automatically. The bounce may be used when a port is required for a special purpose, or to clear the system of all users before stopping the APL\360 operation completely.

If a work session is ended because of failure of the central computer, the active workspace is not stored.

The CONTINUE workspace. When the active workspace is stored automatically, as a result of a disconnect, bounce, or one of the continue commands described below, it goes into the user's private library and is given the name *CONTINUE*. If the active workspace had a password associated with it, *CONTINUE* will be locked with the same password.

If *CONTINUE* is automatically stored, and is not locked, it will be automatically activated at the next sign-on; otherwise, a clear workspace is activated.

Since *CONTINUE* will replace any workspace that may have been previously stored under that name, there is a danger that repeated line failures, while working with a locked workspace, could lead to a complete loss of information. To protect against this possibility, a clear workspace is never stored automatically.

Interrupted activities. An APL operation in progress at the moment of occurrence of a bounce or disconnect may or may not be carried to its normal conclusion. A defined function in progress at such a moment will be suspended, but its progress can be resumed at a later work session in accordance with the procedures given in Part 3. A system command, once begun, will continue to completion regardless of the state of the terminal.

If a bounce or disconnect occurs when the terminal is in definition mode, the definition process is arbitrarily terminated by the system. To proceed with the definition when *CONTINUE* is next activated, the definition mode can be re-established according to the procedures given in Part 3. The continue commands will be rejected in definition mode.

Detailed description. The trouble reports *NO SPACE* and *LIBRARY TABLE FULL* have been omitted from Table 2.1, and are also omitted from the notes below, because their occurrence is infrequent, and no corrective action can be taken from a remote terminal. They can arise in response to a continue command or a save command (see section on library control), and signify that certain of the physical resources of the system have been exhausted.

Elapsed time or time of day, given as a system response, is always in hours, minutes, and seconds; two digits for each, separated by periods. A date response is given as month, day and year; two digits for each, separated by slashes. Clock hours are counted continuously from midnight of the indicated day, and if the system runs past midnight it is possible to have time readings well above 24 hours. For example, 34.22.00 07/11/68 would be 22 minutes past 10 AM on July 12, 1968.

<u>ACTION</u>	<u>NOTES</u>
<u>TCl.</u> <u>Start a work session:</u> This is the <u>sign-on</u> , described in EC3 of Part 1.	See Part 1, EC3.

TC2. End work session:
Enter)OFF
followed by a colon and a
password, if desired.

Passwords longer than eight
characters are accepted, but
only the first eight are
meaningful. Spaces around
the colon are neutral.

Effect:

1. The currently active
workspace will vanish.

There is no effect on any
stored workspace.

2. The duration of the work
session and the amount of
computer time used will be
noted internally for later
accounting.

3. The password, if used,
will become a new lock on
the account number.

Once applied, a lock stays
in effect until explicitly
changed by an ending command
that contains a colon.

An existing lock is removed
if no password follows the
colon.

If a colon is not used, the
existing lock, if any,
remains in force.

4. A dial-up connection to
the central computer will be
broken.

Response:

1. The port number, time of
day, date, and user code
will be printed on one line.

2. Accounting information
will be printed on two
lines, giving terminal
connection time and central
computer time.

Trouble report:

INCORRECT COMMAND

The time used in this
session and cumulative time
since the last accounting
are given in the standard
format, for both terminal
time and computer time.

The DATA light on telephone
datasets will go out.

TC3. End work session and hold dial-up connection:
Enter)OFF HOLD
followed by a colon and a password, if desired.

See note at TC2.

Effect:

1. 2. and 3. Same as for TC2.

4. The dial-up telephone connection will be maintained for 60 seconds, pending a new sign-on.

An attention signal at this time may cause the connection to be broken.

Response:

1. and 2. Same as for TC2.

Trouble report:

INCORRECT COMMAND

TC4. End work session and store active workspace:
Enter)CONTINUE
followed by a colon and a password, if desired.

See note at TC2.

A bounce has the same effect and response as this command. A disconnect has the same effect, but no response.

Effect:

1. A copy of the currently active workspace will be stored in the user's private library with the name CONTINUE. If the workspace had been activated from a stored workspace with a lock, the same lock will be applied to CONTINUE.

This effect will not take place if the active workspace is not holding information.

When the workspace is saved it replaces any workspace previously stored with the name CONTINUE.

2. 3. and 4. Same as for TC2.

Response:

1. Time of day and date, followed by *CONTINUE*.

2. and 3. Same as for TC2, response 1 and 2.

TC5. End work session, store active workspace, and hold dial-up connection:
Enter *)CONTINUE HOLD* followed by a colon and a password, if desired.

Effect:

1. Same as for TC4.

2. and 3. Same as for TC2.

4. Same as for TC3.

Response:

1. 2. and 3. Same as for TC4.

This response will be omitted if the workspace was not saved. See note at Effect 1.

Trouble reports:

NOT WITH OPEN DEFINITION means that the terminal is in definition mode. Close the definition by entering the character ∇. (See mechanics of function definition in Part 3.)

INCORRECT COMMAND

See note at TC2.

Trouble reports:

NOT WITH OPEN DEFINITION
See TC4.

INCORRECT COMMAND

WORKSPACE CONTROL COMMANDS

The commands in this class can replace the active workspace with a clear one, or with a copy of a stored workspace; bring together in the active workspace information from many stored workspaces; form groups within the active workspace; remove unwanted objects from the active workspace; and set controls governing certain operations. No command in this class affects any but the active workspace.

Application packages. The usefulness of a terminal system is enhanced by the availability of many different collections of functions and variables, each of which is organized to satisfy the computational needs of some area of work; for example, standard statistical calculations, exercises for teaching a scholastic subject, complex arithmetic, business accounting, text editing, etc. The workspace-centered organization of APL\360 lends itself to such packaging, because each collection moves as a coherent unit when the workspace containing it is stored or activated.

The copy commands provide a convenient way to assemble packages from components in different workspaces. The group command makes it convenient to have a multiplicity of more specialized packages in a single workspace, sharing common elements, but available individually by copying the appropriate group.

Groups. The group command assigns a single name to a collection of names, in order to provide more convenient reference to selected functions and global variables. The group name can subsequently be used for three purposes: to move a copy of the entire set of referenced objects between workspaces, to incorporate the group members within another group, and to erase, in a single operation, all objects referenced by the group. Each of these is further explained below, in connection with the relevant operation.

Information transfer between workspaces. Information entered or developed within one workspace can be made available within another by means of the copy and protecting-copy commands, which reproduce within the active workspace objects from a stored workspace. These are two

sets of parallel commands which differ only in their treatment of an object in the active workspace which has the same name as an object being reproduced: the copy commands will replace the existing object, whereas the protecting-copy commands will not make the replacement.

A copy command of either type can be applied to an entire workspace or to a single object (i.e., a function, variable, or group). When an entire workspace is copied, all the functions and global variables within it are subject to the operation, but its index origin and output control settings, state indicator, and local variables are left behind.

When a group is copied without protection, both its definition (i.e., the group name and the collection of names composing the group), and the objects referenced by the names within it, are reproduced in the active workspace. When copied with protection, the group itself, or any of the objects referenced by its members, will be omitted in order to protect an object in the active workspace. If the group definition is successfully copied under these circumstances, the names composing it will refer to the global objects by those names in the active workspace, regardless of whether they were copied with the group or present before.

Detailed Description. The term workspace identification is used here to mean either a library number followed by a workspace name, or a workspace name alone. When a name is used alone, the reference is to the user's private library. A key is a colon followed by a password.

<u>ACTION</u>	<u>NOTES</u>
<p><u>WCl. Activate a clear workspace:</u> Enter)CLEAR.</p> <p><u>Effect:</u> 1. A clear workspace will be activated, replacing the presently active workspace.</p>	<p>This command is used to make a fresh start, discarding whatever is in the active workspace.</p> <p>A clear workspace has no variables, groups, or defined functions.</p> <p>Its control settings are: index origin, 1; significant digits, 10; line width, 120.</p> <p>Its workspace identification does not match that of any stored workspace. (See section on library control.)</p>

Response:

1. *CLEAR WS*

WC2. Activate a copy of a stored workspace:

Enter *)LOAD*

followed by a space and a workspace identification (with the key, if required).

Effect:

1. A copy of the designated workspace will be activated, replacing the presently active workspace.

Response:

1. *SAVED*, followed by the time of day and the date that the source workspace was last stored.

WC3. Copy a global object from a stored workspace:

Enter *)COPY*

followed by a space and a workspace identification (with the key, if required), followed by a space and the name of the object to be copied.

Effect:

1. A copy of the designated object will appear in the active workspace with global significance, replacing existing global homonyms.

Trouble report

INCORRECT COMMAND

This command may be used to obtain the use of any workspace in the system whose identification (and password) is known.

Trouble reports

WS NOT FOUND

means there is no stored workspace with the given identification.

WS LOCKED

means that no key, or the wrong key, was used when one was required.

INCORRECT COMMAND

A global object may be a group, function, or global variable.

When applied to a group, all copy commands operate both on the group definition and on objects referenced by the group members.

Members of a group do not necessarily have referents; but a group member without a referent in the source workspace may find one in the active workspace.

Response:

1. *SAVED*, followed by the time of day and the date that the source workspace was last stored.

Trouble reports

NOT WITH OPEN DEFINITION

means that the terminal is in definition mode. Either close the definition by entering ∇ , or defer the copy operation.

WS NOT FOUND

See WC2.

WS LOCKED

See WC2.

OBJECT NOT FOUND

means that the designated workspace does not contain a global object with the given name.

WS FULL

means that the active workspace could not contain all the material requested: if copied at all, a variable will be copied completely; a partially copied function will leave the terminal in definition mode; some objects may be completely overlooked. Status may be determined by using appropriate inquiry commands.

INCORRECT COMMAND

See notes at WC3.

WC3a. Copy all global objects from a stored workspace:

Enter)*COPY*

followed by a space and a workspace identification (with the key, if required).

Effect:

1. A copy of all functions, groups, and global variables in the source workspace will appear in the active workspace with global significance, replacing existing global homonyms.

Local variables, the state indicator, and settings for origin, significant digits, and width are not copied.

Response:

1. *SAVED*, followed by the time of day and the date that the source workspace was last stored.

Trouble reports

NOT WITH OPEN DEFINITION
WS NOT FOUND
WS LOCKED
WS FULL
INCORRECT COMMAND

See WC3 for all meanings.

WC4. Copy a global object from a stored workspace, protecting the active workspace:

Enter *)PCOPY*

followed by a space and a workspace identification (with the key, if required), followed by a space and the name of the object to be copied.

See notes at WC3.

Effect:

1. A copy of the designated object will appear in the active workspace unless there is an existing global homonym.

When a group definition is copied, any member whose referent was blocked will, perforce, refer to the referent of its homonym.

Response:

1. *SAVED*, followed by the time of day and the date that the source workspace was last stored.

2. *NOT COPIED:*, followed by the names of objects not copied, will be printed if appropriate.

Trouble reports

NOT WITH OPEN DEFINITION
WS NOT FOUND
WS LOCKED
OBJECT NOT FOUND
WS FULL
INCORRECT COMMAND

See WC3 for all meanings.

WC4a. Copy all global
objects from a stored
workspace, protecting the
active workspace:

Enter)PCOPY

followed by a space and a
workspace identification
(with the key, if required).

Effect:

1. A copy of all global
objects in the source
workspace which do not have
global homonyms in the
active workspace will appear
in the active workspace.

Response:

1. *SAVED*, followed by the
time of day and the date
that the source workspace
was last stored.

2. *NOT COPIED:*, followed by
the names of objects not
copied, will be printed if
appropriate.

See notes at WC3.

See note at WC3a, Effect 1.

See note at WC4, Effect 1.

Trouble reports

NOT WITH OPEN DEFINITION

WS NOT FOUND

WS LOCKED

WS FULL

INCORRECT COMMAND

See WC3 for all meanings.

WC5. Gather names into a group:

Enter `)GROUP`

followed by a space and one or more names separated by spaces.

Effect:

1. The first name will be the name of a group having the other names as members, subject to the rules given in the adjacent notes. An existing group with the same name will be superseded.

2. If only one name is used in the command, no group is formed, and an existing group by that name is dispersed.

Response: None.

The first name used in the command must not be the name of a function or global variable.

Any name may be a member of a group; names of groups, functions, and global variables, and names without current global referents are all acceptable.

Members may be added to an existing group by using the group name twice in the command: as the first name and as another.

When a group is dispersed the group definition is destroyed, but the referents of the group members are unaffected.

Trouble reports:

`NOT GROUPED, NAME IN USE` means that the first name used in the command is the name of a function or global variable. Erase the offending object, or use a different name.

`INCORRECT COMMAND`

WC6. Erase global objects:
Enter `)ERASE`
followed by a space and the
names of objects to be
deleted, separated by
spaces.

Effect:
1. Named objects having
global significance, other
than pendent functions, will
be expunged.

Response:
`NOT ERASED:`, followed by the
names of functions not
erased will be printed, if
appropriate.

WC7. Set index origin for
array operations: Enter the
characters `)ORIGIN`
followed by a space and a 0
or 1.

Effect:
1. First elements of arrays
in the workspace will be
numbered zero or one, as
indicated, and subsequent
use of index-dependent APL
operations will be
appropriately affected.

Response:
1. `WAS`, followed by the
former origin.

This is the only way to
remove a global variable,
and the most convenient way
to remove a collection of
objects.

Names which do not refer to
global objects are ignored.

When a group is erased, both
the group and the referents
of its members are expunged.

Trouble report:
`INCORRECT COMMAND`

A dynamically executable
equivalent function is
available (see Part 4).

These matters are explained
in Part 3.

Trouble report:
`INCORRECT COMMAND`

WC8. Set maximum for significant digits in output:
Enter)*DIGITS*
followed by a space and an integer between 1 and 16 inclusive.

Effect:
1. Subsequent output of numbers will show no greater number of significant digits than indicated.

Response:
1. *WAS*, followed by the former maximum.

WC9. Set maximum width for an output line:
Enter)*WIDTH*
followed by a space and an integer between 30 and 130 inclusive.

Effect:
1. Subsequent output of all kinds, except messages between terminals, will be limited to a line width no greater than the number of spaces indicated.

Response:
1. *WAS*, followed by the former maximum width.

A dynamically executable equivalent function is available (see Part 4).

This command has no effect on the precision of internal calculations, which is approximately 16 decimal digits.

Trouble report:
INCORRECT COMMAND

A dynamically executable equivalent function is available (see Part 4).

This affects neither the mechanical margin stops nor the allowable length of input lines.

Trouble report:
INCORRECT COMMAND

WC10. Change workspace identification:
Enter)*WSID*
followed by a space and a workspace identification.

This command can be used to guard against inadvertently changing a stored workspace that has just been loaded; and conversely, to enable the replacement of a stored workspace without first using the drop command, when the active workspace came from a different source. (See section on library control commands.)

Effect:

1. The active workspace will assume the specified identification. A lock associated with the workspace will be retained.

See command LC1 for the implications of this.

Response:

1. *WAS*, followed by the former workspace identification.

Trouble report:

INCORRECT COMMAND

LIBRARY CONTROL COMMANDS

There are two basic operations performed by the commands in this class. The save commands cause a copy of an active workspace to be stored in a library, and the drop command causes such a stored copy to be destroyed.

The save commands and the load command are symmetric, in the sense that a load command destroys an active workspace by replacing it with a copy of a stored workspace, while a save command may destroy a stored workspace by replacing it with a copy of the active workspace.

Continuity of work. When a workspace is stored, an exact copy of the active workspace is made, including the state indicator and intermediate results from the partial execution of halted functions. These functions can be restarted without loss of continuity (see Part 3), which permits considerable flexibility in planning use of the

system. For example, lengthy calculations do not have to be completed at one terminal session; student work can be conducted over a series of short work periods, to suit class schedules; and mathematical experimentation or the exploration of system models can be done over long periods of time, at the investigator's convenience.

Workspace identification. A library number and a name, together, uniquely identify each stored workspace in the system. An active workspace is also identified by a library number and a name, and as copies of stored workspaces are activated, or copies of the active workspace are stored, the identification of the active workspace may change according to the following rules:

1. A workspace activated from a library assumes the identification of its source.
2. When a copy of the active workspace is stored, the active workspace assumes the identification assigned to the stored copy.
3. The library number and name may be arbitrarily changed by the use of command `WC10`.
4. A clear workspace activated by a clear command, a sign-on, or a system failure is called *CLEAR WS*, which cannot be the name of a stored workspace.

The identification of active workspaces is used in two ways. First, as a safeguard against the inadvertent replacement of a stored workspace by an unrelated one: an attempt to replace, by a copy of the active workspace, any stored workspace other than the one with the same identification (or the one named *CONTINUE*), will be stopped. Second, as a convenience when the active workspace is to be re-stored with changes: the use of the command `)SAVE`, without modification, implicitly uses the identification of the active workspace.

Library and account numbers. A user's account number is also the number of his private library. The numbers of public libraries range from 1 to 999, and do not correspond to any account number.

Each stored workspace has implicitly associated with it the account number signed on at the terminal from which the save command was entered, and may not be either replaced or erased, except from a terminal signed on with the same account number. Thus, a user is prevented from affecting

the state of another user's private library, or tampering with public library workspaces which he did not store. He may, of course, activate a copy of any workspace stored in the system, if he knows the library number and name (and password, if required).

Storage allotment. A user of APL\360 is assigned library space in terms of the maximum number of stored workspaces he may have at one time. This quota applies to the combined total of workspaces stored either in his private library or in public libraries. The allotment for each user is determined by those responsible for the general management of a particular system, and can be changed from the recording terminal, as required, within the bounds of the physical resources of the system.

Up to the number in his quota, a user may assign arbitrary names to the workspaces he stores. Beyond that point he always has available one workspace named *CONTINUE* in his private library.

Use of the *CONTINUE* workspace. This workspace has the property that it may be freely replaced by an active workspace having any identification whatsoever. It is thus always available as temporary storage, but carries with it the danger of being easily replaced, as described in the section on terminal control commands.

The attributes of the *CONTINUE* workspace are the same whether stored as a result of a continue command, disconnect, or bounce, or stored by virtue of a save command using that name. In the last case, the active workspace assumes the name *CONTINUE*, as it would any other name under like circumstances.

Purging a workspace. The sequence of commands, `)SAVE ABC123,)CLEAR,)COPY ABC123`, will purge the active workspace, clearing it of all but its functions, groups, and global variables, and reset its controls (see WCl). This often results in more usable space than can otherwise be realized. Subsequently, the commands `)WSID ABC123` and `)SAVE` may be used to store a copy of the purged workspace under its former name.

Detailed Description. The term workspace identification will be used with the same significance as for the workspace control commands.

ACTION

NOTES

LCl. Re-store a copy of the active workspace:
Enter)*SAVE*

Effect:

1. A copy of the active workspace will replace the stored workspace with the same identification.

New workspaces can be stored by this command only if the identification of the active workspace has been changed by WC10.

2. A password associated with the active workspace will continue in effect, and the stored workspace will be locked with this password.

This forestalls inadvertent omission of a lock while actively engaged with a confidential workspace.

Response:

1. The time of day, date, and workspace identification will be printed.

Trouble reports:

NOT WITH OPEN DEFINITION means that the terminal is in function definition mode. Either close the definition by entering ∇, or defer the save operation.

NOT SAVED, WS QUOTA USED UP means that the allotted number of stored workspaces has previously been reached. Unless this is increased, the workspace can be stored only by replacing a workspace already stored. *CONTINUE* may be replaced directly; any other must be erased first, or the identification of the active workspace must be made to match by WC10.

NOT SAVED, THIS WS IS
CLEAR WS
results from the fact that
CLEAR WS cannot be the name
of a stored workspace.
Either change the name by
WC10, or use LC1a.

IMPROPER LIBRARY REFERENCE
means that an attempt was
made either to replace a
stored workspace that is not
under control of the account
number signed on at the
terminal, or to store into a
non-existent library.

INCORRECT COMMAND

LC1a. Store a copy of the
active workspace:
Enter)SAVE
followed by a space and a
workspace identification,
with a colon and password,
if desired.

Effect:

1. A copy of the active
workspace will be stored
with the designated
identification, and with the
assigned lock, if a password
was used.

2. The active workspace will
assume the workspace
identification used in the
command.

This form of the save
command allows new
workspaces to be added to a
library more conveniently,
and permits locks to be
added or removed from
workspaces already present.

A stored workspace with the
same identification will be
replaced.

A lock on a stored workspace
will not be retained if the
command does not include a
lock explicitly.

To this extent only, this
command may affect the state
of the active workspace.

Response:

1. The time of day and date will be printed.

Trouble reports:

NOT WITH OPEN DEFINITION
means the same as for LC1.

NOT SAVED, WS QUOTA USED UP
means the same as for LC1.

NOT SAVED, THIS WS IS
followed by identification of the active workspace, means a stored workspace with the identification used in the command exists, but this identification does not match that of the active workspace.

IMPROPER LIBRARY REFERENCE
means the same as for LC1.

INCORRECT COMMAND

LC2. Erase a stored workspace:

Enter *)DROP*

followed by a space and a workspace identification.

Since a key is not used, a locked workspace whose key has been lost can always be removed from the system.

Effect:

1. The designated stored workspace will be expunged.

This command has no effect on the active workspace, regardless of its identification.

Response:

1. The time of day and date will be printed.

Trouble reports:

IMPROPER LIBRARY REFERENCE
means that an attempt was made to drop a workspace stored by another user.

WS NOT FOUND
means that there is no stored workspace with the identification used in the command.

INCORRECT COMMAND

INQUIRY COMMANDS

Most of the commands in this class concern the state of the active workspace. Of the others, one command lists the names of workspaces in libraries, and two commands are useful for locating another user at a connected terminal, in order to communicate with him.

User codes. The communication commands described in the next section require that the port number of the person to be addressed be known. The inquiry commands that provide this information operate through the device of user codes, which serve within the system as partial identification of users. (The user account numbers, which completely identify users within the system, are not used for this purpose, and are treated as private information.) A user code comprises the first three characters of his name, as it appears in the sign-on response (Part 1, EC3, Response 2).

A user code is considered to be only partial identification because it may not be unique. Therefore, these commands should be used advisedly: before addressing substantive messages to a terminal which has been identified by a user code, further confirmation of the receiver's identity should be sought.

Detailed description.

ACTION

NOTES

IQ1. List names of defined functions:

Enter)*FNS*

followed by an alphabetic character, if desired.

Effect: None.

Response:

1. The names of defined functions in the active workspace will be printed alphabetically, starting with the specified letter. If a letter was not used, all function names will be listed.

Trouble report

INCORRECT COMMAND

IQ2. List names of global variables:

Enter)*VAR*S

followed by an alphabetic character, if desired.

Effect: None.

Response:

1. The names of global variables in the active workspace will be printed alphabetically, starting with the specified letter. If a letter was not used, all names of global variables will be listed.

Trouble report

INCORRECT COMMAND

IQ3. List names of groups:

Enter)*GR*PS

followed by an alphabetic character, if desired.

Effect: None.

Response:

1. The names of groups in the active workspace will be printed alphabetically, starting with the letter used. If a letter was not used, all group names will be listed.

Trouble report

INCORRECT COMMAND

IQ4. List membership of designated group:

Enter)GRP

followed by the name of the group.

Effect: None.

Response:

1. The names in the group will be printed.

There will be no response if there is no group with the designated name in the active workspace.

Trouble report

INCORRECT COMMAND

IQ5. List halted functions:

Enter)SI

The line numbers on which halted functions have stopped are available for dynamic use through the system-dependent functions I26 and I27. (See Part 3.)

Effect: None.

Response:

1. The names of halted functions will be listed, most recent ones first. With each name will be given the line number on which execution stopped. Suspend- ed functions will be distinguished from pendent functions by an asterisk.

This display is the state indicator; its significance and use is explained in Part 3.

Trouble report

INCORRECT COMMAND

IQ6. List halted functions with names of local variables:
Enter)SIV

Effect: None.

Response:

1. The response will be the same as for IQ5, except that with each function listed there will appear a listing of its local variables.

Trouble report
INCORRECT COMMAND

IQ7. Give identification of active workspace:
Enter)WSID

Effect: None.

Response:

1. The identification of the active workspace will be printed. The library number will be included only if it differs from the account number associated with the terminal.

Trouble report
INCORRECT COMMAND

IQ8. List names of stored workspaces:
Enter)LIB
followed, if necessary, by a library number.

Effect: None.

Response:

1. The names of workspaces in the designated library will be printed. If no number was used, the account number associated with the terminal will be taken as the library number.

A library number is not required for listings of the user's private library.

Trouble reports

IMPROPER LIBRARY REFERENCE means that an attempt was made to obtain a listing of another user's private library, or of a non-existent library.

INCORRECT COMMAND

IQ9. List ports in use and codes of connected users:
Enter)*PORTS*

Effect: None.

Response:
1. Port numbers in use will be printed with the associated user code.

Trouble report
INCORRECT COMMAND

IQ10. List port numbers associated with designated user code:
Enter)*PORTS*
followed by the user code.

User codes are not necessarily unique, and the information derived from this command and IQ9 should be used advisedly.

Effect: None.

Response:
1. The port numbers of connected users identified by the code will be printed.

Trouble report
INCORRECT COMMAND

COMMUNICATION COMMANDS

There are two pairs of commands in this class. One pair addresses any connected terminal, and one pair addresses only the system recording terminal.

A message can be received by a terminal only when its keyboard is locked, and except for public address announcements from the system recording terminal, only if it is also not in the process of function execution. Hence, to facilitate two-way communication, one of each pair of communication commands results in locking the keyboard of the sending terminal, pending the receipt of a reply. A keyboard so locked can be unlocked by an attention signal.

Incoming messages from the system recording terminal are prefixed by *OPR:*, and public address messages are prefixed by *PA!:*. The latter will interrupt the execution of defined functions, but the point of suspension will be printed, and execution can be resumed without loss of continuity, as explained in Part 3.

If the interaction at a terminal must be interrupted for a prolonged period while the terminal is still connected, it is good practice to lock the keyboard so that a message may be received. This can be done by addressing a message of the proper type to the terminal's own port number.

Detailed description. The length of a message is restricted to a single line, not exceeding 120 characters in length. However, messages are not subject to the width settings of either the sending or receiving terminal.

ACTION

NOTES

CMI. Address text to designated port:
Enter)*MSGN*
followed by a port number and any one-line text.

A message addressed to an unused or non-existent port will be reflected back to the sending terminal, which then plays the role of both sender and receiver.

Effect:

1. The keyboard will lock while the text is being transmitted.
2. The text will be printed at the receiving terminal, prefixed by the port number of the sending terminal.
3. The keyboard will unlock when the transmission is completed.

Response:

1. *SENT*

Trouble reports

MESSAGE LOST

means just that. It happens when attention is signalled before a message is delivered, or an equivalent transmission disturbance occurs.

INCORRECT COMMAND

CM2. Address text to designated port and lock keyboard:

Enter)MSG
followed by a port number
and any one-line text.

Effect:

1. Same as CM1 effect 1.
2. Same as CM1, Effect 2, except for a prefix R, to indicate that a reply is awaited.
3. The keyboard will remain locked after the response is printed.

Response:

1. SENT

CM3. Address text to system recording terminal:

Enter)OPRN
followed by any one-line
text.

Effect:

1. 2. and 3. Same as CM1.

Response:

1. SENT

See note at CM1.

The keyboard can be unlocked, before receiving a reply, by means of an attention signal.

Trouble reports

MESSAGE LOST

See CM1.

INCORRECT COMMAND

See note at CM1.

Trouble reports

MESSAGE LOST

See CM1.

INCORRECT COMMAND

CM4. Address text to system
recording terminal and lock
keyboard:

Enter)*OPR*
followed by any one-line
text.

See note at CM1.

Effect:

1. 2. and 3. Same as CM2.

Response:

1. *SENT*

Trouble reports

MESSAGE LOST

See CM1.

INCORRECT COMMAND



PART 3

THE LANGUAGE

The APL\ 360 Terminal System executes system commands or mathematical statements entered on a terminal typewriter. The system commands were treated in Part 2; the mathematical statements will be treated here.

Acceptable statements may employ either primitive functions (e.g. + - × ÷) which are provided by the system, or defined functions, which the user provides by entering their definitions on the terminal.

If system commands are not used, the worst that can possibly result from erroneous use of the keyboard is the printing of an error report. It is therefore advantageous to experiment freely and to use the system itself for settling any doubts about its behavior. For example, to find what happens in an attempted division by zero, simply enter the expression $4 \div 0$. If ever the system seems unusually slow to respond, execute an attention signal to interrupt execution and unlock the keyboard.

The Sample Terminal Session of Appendix A shows actual intercourse with the system which may be used as a model in gaining facility with the terminal. The examples follow the text and may well be studied concurrently. More advanced programming examples appear in Appendix B.

The primitive functions and the defined functions available in libraries can be used without knowledge of the means of defining functions. These means are treated in the four contiguous sections beginning with Defined Functions and ending with Homonyms. These sections may be skipped without loss of continuity.

FUNDAMENTALS

Statements. Statements are of two main types, the branch (denoted by \rightarrow and treated in the section on Defined Functions), and the specification. A typical specification statement is of the form

$$X \leftarrow 3 \times 4$$

This statement assigns to the variable X the value resulting from the expression to the right of the specification arrow.

If the variable name and arrow are omitted, the resulting value is printed. For example:

3×4
12

Results typed by the system begin at the left margin whereas entries from the keyboard are automatically indented. The keyboard arrangement is shown in Figure 1.2.

Scalar and vector constants. All numbers entered via the keyboard or typed out by the system are in decimal, either in conventional form (including a decimal point if appropriate) or in exponential form. The exponential form consists of an integer or decimal fraction followed immediately by the symbol *E* followed immediately by an integer. The integer following the *E* specifies the power of ten by which the part preceding the *E* is to be multiplied. Thus $1.44E2$ is equivalent to 144.

Negative numbers are represented by a negative sign immediately preceding the number, e.g., -1.44 and $-144E2$ are equivalent negative numbers. The negative sign can be used only as part of a constant and is to be distinguished from the negation function which is denoted, as usual, by the minus sign $-$.

A constant vector is entered by typing the constant components in order, separated by one or more spaces. A character constant is entered by typing the character between quotation marks, and a sequence of characters entered in quotes represents a vector whose successive components are the characters themselves. Such a vector is printed by the system as the sequence of characters, with no enclosing quotes and with no separation of the successive elements. The quote character itself must be typed in as a pair of quotes. Thus, the abbreviation of *CANNOT* is entered as `'CAN''T'` and prints as *CAN'T*.

Names and Spaces. As noted in Part 2, the name of a variable or defined function may be any sequence of letters or digits beginning with a letter and not containing a space. A letter may be any of the characters *A* to *Z*, or any one of these characters underscored, e.g., *A* or *B*.

Spaces are not required between primitive functions and constants or variables, or between a succession of primitive functions, but they may be used if desired. Spaces are needed to separate names of adjacent defined functions, constants, and variables. For example, the expression $3+4$ may be entered with no spaces, but if *F* is a defined

function, then the expression $3 \ F \ 4$ must be entered with the indicated spaces. The exact number of spaces used in succession is of no importance and extra spaces may be used freely.

Overstriking and erasure. Backspacing serves only to position the carriage and does not cause erasure or deletion of characters. It can be used:

1. to insert missing characters (such as parentheses) if space has previously been left for them,
2. to form compound characters by overstriking (e.g. ϕ and $!)$, and
3. to position the carriage for erasure, which is effected by striking the linefeed (marked ATTN on IBM 2741 terminals). The linefeed has the effect of erasing the character at the position of the carriage, and all characters to the right.

End of Statement. The end of a statement is indicated by striking the carriage return (followed, on some terminals, by an explicit transmission signal as described in Part 1). The typed entry is then interpreted exactly as it appears on the page, regardless of the time sequence in which the characters were typed.

Order of execution. In a compound expression such as $3 \times 4 + 6 \div 2$, the functions are executed (evaluated) from rightmost to leftmost, regardless of the particular functions appearing in the expression. (The foregoing expression evaluates to 21.) When parentheses are used, as in the expression $W \leftarrow (3 \uparrow Q) \div X \times Y - Z$, the same rule applies, but, as usual, an enclosed expression must be completely evaluated before its results can be used. Thus, the foregoing expression is equivalent to $W \leftarrow (3 \uparrow Q) \div (X \times (Y - Z))$.

In general, the rule can be expressed as follows: every function takes as its righthand argument the entire expression to its right, up to the right parenthesis of the pair that encloses it.

Error reports. The attempt to execute an invalid statement will cause one of the error reports of Table 3.1 to be typed out. The error report will be followed by the offending statement with a caret typed under the point in the statement where the error was detected. If the caret lies to the right of a specification arrow, the specification has not yet been performed.

TYPE	Cause; CORRECTIVE ACTION
CHARACTER	Illegitimate overstrike.
DEPTH	Excessive depth of function execution. CLEAR STATE INDICATOR.
DOMAIN	Arguments not in the domain of the function.
DEFN	Misuse of ∇ or □ symbols: 1. ∇ is in some position other than the first. 2. The function is pendent. DISPLAY STATE INDICATOR AND CLEAR AS REQUIRED. 3. Use of other than the function name alone in reopening a definition. 4. Improper request for a line edit or display.
INDEX	Index value out of range.
LABEL	Name of already defined function used as a label, or colon used other than in function definition and between label and statement.
LENGTH	Shapes not conformable.
RANK	Ranks not conformable.
RESEND	Transmission failure. RE-ENTER. IF CHRONIC, REDIAL OR HAVE TERMINAL OR PHONE REPAIRED.
SYNTAX	Invalid syntax; e.g., two variables juxtaposed; function used without appropriate arguments as dictated by its header; unmatched parentheses.
SYMBOL TABLE FULL	Too many names used. ERASE SOME FUNCTIONS OR VARIABLES, THEN SAVE, CLEAR, AND COPY.
SYSTEM	Fault in internal operation of APL\ 360. RELOAD OR SAVE, CLEAR, AND COPY. SEND TYPED RECORD, INCLUDING ALL WORK LEADING TO THE ERROR, TO THE SYSTEM MANAGER.
VALUE	Use of name which has not been assigned a value. ASSIGN A VALUE TO THE VARIABLE, OR DEFINE THE FUNCTION.
WS FULL	Workspace is filled (perhaps by temporary values produced in evaluating a compound expression). CLEAR STATE INDICATOR, ERASE NEEDLESS OBJECTS, OR REVISE CALCULATIONS TO USE LESS SPACE.

Table 3.1 ERROR REPORTS

If an invalid statement is encountered during execution of a defined function, the error report includes the function name and the line number of the invalid statement. The recommended procedure at this point is to enter a right arrow (\rightarrow) alone, and then retry with an amended statement. The matter is treated more fully in the section on Suspended Function Execution.

Names of primitive functions. The primitive functions of the language are summarized in Tables 3.2 and 3.8, and will be discussed individually in subsequent sections. The tables show one suggested name for each function. This is not intended to discourage the common mathematical practice of vocalizing a function in a variety of ways (for example, $X \div Y$ may be expressed as "X divided by Y", or "X over Y"). Thus, the expression ρ^M yields the dimension of the array M , but the terms size or shape may be preferred both for their brevity and for the fact that they avoid potential confusion with the dimensionality or rank of the array.

The importance of such names and synonyms diminishes with familiarity. The usual tendency is toward the use of the name of the symbol itself (e.g., "rho" (ρ) for "size", and "iota" (ι) for "index generator"), probably to avoid unwanted connotations of any of the chosen names.

SCALAR FUNCTIONS

Each of the primitive functions is classified as either scalar or mixed. Scalar functions are defined on scalar (i.e., individual) arguments and are extended to arrays in four ways: element-by-element, reduction, inner product, and outer product, as described in the section on Functions on Arrays. Mixed functions are discussed in a later section.

The scalar functions are summarized in Table 3.2. Each is defined on real numbers or, as in the case of the logical functions and and or, on some subset of them. No functional distinction is made between "fixed point" and "floating point" numbers, this being primarily a matter of the representation in a particular medium, and the user of the terminal system need have no concern with such questions unless his work strains the capacity of the machine with respect to either space or accuracy.

Three different representations for numbers are used internally, and transformations among them are carried out automatically. Integers less than 2 to the power 52 are carried with full precision; larger numbers and non-integers are carried to a precision of about 16 decimal digits.



Monadic form fB		f	Dyadic form AfB																															
Definition or example	Name		Name	Definition or example																														
$+B \leftrightarrow 0+B$	Plus	+	Plus	$2+3.2 \leftrightarrow 5.2$																														
$-B \leftrightarrow 0-B$	Negative	-	Minus	$2-3.2 \leftrightarrow -1.2$																														
$\times B \leftrightarrow (B>0)-(B<0)$	Signum	\times	Times	$2\times 3.2 \leftrightarrow 6.4$																														
$\div B \leftrightarrow 1\div B$	Reciprocal	\div	Divide	$2\div 3.2 \leftrightarrow 0.625$																														
$\frac{B}{\lceil 3.14 \rceil} \mid \lceil B \rceil \mid \lfloor B \rfloor$	Ceiling	\lceil	Maximum	$3\lceil 7 \rceil \leftrightarrow 7$																														
$\frac{B}{\lfloor -3.14 \rfloor} \mid \lfloor B \rfloor \mid \lceil B \rceil$	Floor	\lfloor	Minimum	$3\lfloor 7 \rfloor \leftrightarrow 3$																														
$*B \leftrightarrow (2.71828\dots)*B$	Exponential	*	Power	$2*3 \leftrightarrow 8$																														
$\odot *N \leftrightarrow N \leftrightarrow * \odot N$	Natural logarithm	\odot	Logarithm	$A \odot B \leftrightarrow \text{Log } B \text{ base } A$ $A \odot B \leftrightarrow (*B) \div \odot A$																														
$\lvert^{-} 3.14 \leftrightarrow 3.14$	Magnitude	\lvert	Residue	<table border="1"> <thead> <tr> <th>Case</th> <th>$A \lvert B$</th> </tr> </thead> <tbody> <tr> <td>$A \neq 0$</td> <td>$B - (A \times \lfloor B \div A \rfloor)$</td> </tr> <tr> <td>$A = 0, B \geq 0$</td> <td>$B$</td> </tr> <tr> <td>$A = 0, B < 0$</td> <td>Domain error</td> </tr> </tbody> </table>	Case	$A \lvert B$	$A \neq 0$	$B - (A \times \lfloor B \div A \rfloor)$	$A = 0, B \geq 0$	B	$A = 0, B < 0$	Domain error																						
Case	$A \lvert B$																																	
$A \neq 0$	$B - (A \times \lfloor B \div A \rfloor)$																																	
$A = 0, B \geq 0$	B																																	
$A = 0, B < 0$	Domain error																																	
$!0 \leftrightarrow 1$ $!B \leftrightarrow B \times !B - 1$ or $!B \leftrightarrow \text{Gamma}(B+1)$	Factorial	!	Binomial coefficient	$A!B \leftrightarrow (!B) \div (!A) \times !B - A$ $2!5 \leftrightarrow 10 \quad 3!5 \leftrightarrow 10$																														
$?B \leftrightarrow \text{Random choice from } {}_1B$	Roll	?	Deal	A Mixed Function (See Table 3.8)																														
$\circ B \leftrightarrow B \times 3.14159\dots$	Pi times	\circ	Circular	See Table at left																														
$\sim 1 \leftrightarrow 0 \quad \sim 0 \leftrightarrow 1$	Not	\sim																																
<table border="1"> <thead> <tr> <th>$(-A) \circ B$</th> <th>A</th> <th>$A \circ B$</th> </tr> </thead> <tbody> <tr> <td>$(1-B*2)*.5$</td> <td>0</td> <td>$(1-B*2)*.5$</td> </tr> <tr> <td>Arcsin B</td> <td>1</td> <td>Sine B</td> </tr> <tr> <td>Arccos B</td> <td>2</td> <td>Cosine B</td> </tr> <tr> <td>Arctan B</td> <td>3</td> <td>Tangent B</td> </tr> <tr> <td>$(-1+B*2)*.5$</td> <td>4</td> <td>$(1+B*2)*.5$</td> </tr> <tr> <td>Arcsinh B</td> <td>5</td> <td>Sinh B</td> </tr> <tr> <td>Arccosh B</td> <td>6</td> <td>Cosh B</td> </tr> <tr> <td>Arctanh B</td> <td>7</td> <td>Tanh B</td> </tr> </tbody> </table>		$(-A) \circ B$	A	$A \circ B$	$(1-B*2)*.5$	0	$(1-B*2)*.5$	Arcsin B	1	Sine B	Arccos B	2	Cosine B	Arctan B	3	Tangent B	$(-1+B*2)*.5$	4	$(1+B*2)*.5$	Arcsinh B	5	Sinh B	Arccosh B	6	Cosh B	Arctanh B	7	Tanh B						
$(-A) \circ B$	A	$A \circ B$																																
$(1-B*2)*.5$	0	$(1-B*2)*.5$																																
Arcsin B	1	Sine B																																
Arccos B	2	Cosine B																																
Arctan B	3	Tangent B																																
$(-1+B*2)*.5$	4	$(1+B*2)*.5$																																
Arcsinh B	5	Sinh B																																
Arccosh B	6	Cosh B																																
Arctanh B	7	Tanh B																																
Table of Dyadic \circ Functions			\wedge And \vee Or ∇ Nand ∇ Nor $<$ Less \leq Not greater $=$ Equal \geq Not less $>$ Greater \neq Not Equal	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>$A \wedge B$</th> <th>$A \vee B$</th> <th>$A \nabla B$</th> <th>$A \nabla B$</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>1</td> <td>0</td> <td>0</td> </tr> </tbody> </table> Relations Result is 1 if the relation holds, 0 if it does not: $3 \leq 7 \leftrightarrow 1$ $7 \leq 3 \leftrightarrow 0$	A	B	$A \wedge B$	$A \vee B$	$A \nabla B$	$A \nabla B$	0	0	0	0	1	1	0	1	0	1	1	0	1	0	0	1	1	0	1	1	1	1	0	0
A	B	$A \wedge B$	$A \vee B$	$A \nabla B$	$A \nabla B$																													
0	0	0	0	1	1																													
0	1	0	1	1	0																													
1	0	0	1	1	0																													
1	1	1	1	0	0																													

Table 3.2: PRIMITIVE SCALAR FUNCTIONS

For operations such as floor and ceiling, and in comparisons, a "fuzz" of about $1E^{-13}$ is applied in order to avoid anomalous results that might otherwise be engendered by doing decimal arithmetic on a binary machine.

Two of the functions of Table 3.2, the relations \neq and $=$, are defined on characters as well as on numbers.

Monadic and dyadic functions. Each of the functions defined in Table 3.2 may be used in the same manner as the familiar arithmetic functions $+$ $-$ \times and \div . Most of the symbols employed may denote either a monadic function (which takes one argument) or a dyadic function (which takes two arguments). For example, $\lceil Y$ denotes the monadic function ceiling applied to the single argument Y , and $X \lceil Y$ denotes the dyadic function maximum applied to the two arguments X and Y . Any such symbol always denotes a dyadic function if possible, i.e., it will take a left argument if one is present.

At this point it may be helpful to scrutinize each of the functions of Table 3.2 and to work out some examples of each, either by hand or on a terminal. However, it is not essential to grasp all of the more advanced mathematical functions (such as the hyperbolic functions \sinh , \cosh , and \tanh , or the extension of the factorial to non-integer arguments) in order to proceed. Treatments of these functions are readily available in standard texts.

Certain of the scalar functions deserve brief comment. The residue function $A|B$ has the usual definition of residue used in number theory. For positive integer arguments this is equivalent to the remainder obtained on dividing B by A , and may be stated more generally as the smallest non-negative member of the set $B - N \times A$, where N is any integer.

This formulation covers the case of a zero left argument as shown in Table 3.2. The conventional definition is extended in two further respects:

1. The left argument A need not be positive; however, the value of the result depends only on the magnitude of A .
2. The arguments need not be integral. For example, $1|2.6$ is 0.6 and $1.5|8$ is 0.5.

The expression $\sqrt[8]{*}.5$ (square root of $\sqrt[8]{*}$) yields a domain error, but $\sqrt[8]{*}1\div 3$ has the value $\sqrt[8]{*}2$. More generally, $A*B$ is valid for $A < 0$ if the right argument is (a close approximation to) a rational number with an odd denominator not greater than 85.

The factorial function $N!$ is defined in the usual way as the product of the first N positive integers. It is also extended to non-integer values of the argument N and is equivalent to the Gamma function of $N+1$.

The function $A!B$ (pronounced A out of B) is defined as $(B!) \div (A!) \times B - A$. For integer values of A and B , this is the number of combinations of B things taken A at a time. (It is related to the Complete Beta function as follows: $\text{Beta}(P, Q) \leftrightarrow \frac{1}{Q} \times (P-1)! P + Q - 1$.)

The symbols $< \leq = \geq >$ and \neq denote the relations less than, less than or equal, etc., in the usual manner. However, an expression of the form $A < B$ is treated not as an assertion, but as a function which yields a 1 if the proposition is true, and 0 if it is false. For example:

```

      3 ≤ 7
1     7 ≤ 3
0

```

When applied to logical arguments (i.e., arguments whose values are limited to 0 and 1), the six relations are equivalent to six of the logical functions of two arguments. For example, \leq is equivalent to material implication, and \neq is equivalent to exclusive-or. These six functions together with the and, or, nand, and nor shown in Table 3.2 exhaust the nontrivial logical functions of two logical arguments.

Vectors. Each of the monadic functions of Table 3.2 applies to a vector, element by element. Each of the dyadic functions applies element by element to a pair of vectors of equal dimension or to one scalar and a vector of any dimension, the scalar being used with each component of the vector. For example:

```

      1 2 3 4×4 3 2 1
4  6  6  4
      2+1 2 3 4
3  4  5  6
      1 2 3 4[2
2  2  3  4

```

Index generator. If N is a non-negative integer, then ${}_1N$ denotes a vector of the first N integers. The dimension of the vector ${}_1N$ is therefore N ; in particular, ${}_11$ is a vector of length one which has the value 1, and ${}_10$ is a vector of

dimension zero, also called an empty vector. The empty vector prints as a blank. For example:

```

      1 4
1 2 3 4
      1 5
1 2 3 4 5
      1 0

```

Empty vector prints as a blank

```

      6-1 6
5 4 3 2 1 0
      2x1 0

```

Scalar applies to all (i.e., 0) elements of 10, resulting in an empty vector

```

      2x1 6
2 4 6 8 10 12

```

The index generator is one of the class of mixed functions to be treated in detail later; it is included here because it is useful in examples.

DEFINED FUNCTIONS

Introduction. It would be impracticable and confusing to attempt to include as primitives in a language all of the functions which might prove useful in diverse areas of application. On the other hand, in any particular application there are many functions of general utility whose use should be made as convenient as possible. This need is met by the ability to define and name new functions, which can then be used with the convenience of primitives.

This section introduces the basic notions of function definition and illustrates the use of defined functions. Most of the detailed mechanics of function definition, revision, and display, are deferred to the succeeding section.

The sequence

```

      ∇ SPHERE
[1] SURF ← 4 × 3.14159 × R × R
[2] VOL ← SURF × R ÷ 3
[3] ∇

```

is called a function definition; the first ∇ (pronounced del) marks the beginning of the definition and the second ∇ marks the conclusion: the name following the first ∇ (in this case *SPHERE*) is the name of the function defined, the numbers in brackets are statement numbers, and the accompanying statements form the body of the function definition.

The act of defining a function neither executes nor checks for validity the statements in the body; what it does is make the function name thereafter equivalent to the body. For example:

	$\forall SPHERE$	Definition of the
[1]	$SURF \leftarrow 4 \times 3.14159 \times R \times R$	function $SPHERE$
[2]	$VOL \leftarrow SURF \times R \div 3$	
[3]	∇	
	$R \leftarrow 2$	Specification and display
	R	of the argument R
2		
	$SURF$	$SURF$ has not been
VALUE ERROR		assigned a value
	$SURF$	
	\wedge	
	$SPHERE$	Execution of $SPHERE$
	$SURF$	$SURF$ and VOL now have
50.26544		values assigned by the
	VOL	execution of $SPHERE$
33.51029333		
	$R \leftarrow 1$	Use of $SPHERE$ for
	$SPHERE$	a new value of the
	$SURF$	argument R
12.56636		
	VOL	
4.188786667		

Branching. Statements in a function are normally executed in the order indicated by the statement numbers, and execution terminates at the end of the last statement in the sequence. This normal order can be modified by branches. Branches make possible the construction of iterative procedures.

The expression $\rightarrow 4$ denotes a branch to statement 4 and causes statement 4 of the function to be executed next. In general, the arrow may be followed by any expression which, to be effective, must evaluate to an integer. This value is the number of the statement to be executed next. If the integer lies outside the range of statement numbers of the body of the function, the branch ends the execution of the function.

If the value of the expression to the right of a branch arrow is a non-empty vector, the branch is determined by its first component. If the vector is empty (i.e., of zero dimension) the branch is vacuous and the normal sequence is followed.

The following examples illustrate various methods of branching used in three equivalent functions (*SUM*, *SUM1*, and *SUM2*) for determining *S* as the sum of the first *N* integers:

```

    ∇SUM
[1]  S←0
[2]  I←1
[3]  →4×I≤N          Branch to 4×1 (i.e., 4) or to 4×0 (out)
[4]  S←S+I
[5]  I←I+1
[6]  →3              Unconditional branch to 3
[7]  ∇
    N←1
    SUM
    S
1
    N←2
    SUM
    S
3
    N←5
    SUM
    S
15
    ∇SUM1            Equivalent to SUM
[1]  S←0
[2]  I←1
[3]  →0×I>N        Branch to 0(out) or continue to next
[4]  S←S+I          line since 0×10 is an empty vector
[5]  I←I+1
[6]  →3              Unconditional branch to 3
[7]  ∇
    N←5
    SUM1
    S
15
    ∇SUM2            Equivalent to SUM
[1]  S←0
[2]  I←0
[3]  S←S+I
[4]  I←I+1
[5]  →3×I≤N        Branch to 3 or fall through(and out)
[6]  ∇

```

From the last two functions in the foregoing example, it should be clear that the expression \times_1 occurring in a branch may often be read as "if". For example, $\rightarrow 3 \times_1 I \leq N$ may be read as "Branch to 3 if *I* is less than or equal to *N*."

Local and global variables. A variable is normally global in the sense that its name has the same significance no matter what function or functions it may be used in. However, the iteration counter I occurring in the foregoing function SUM is of interest only during execution of the function; it is frequently convenient to make such a variable local to a function in the sense that it has meaning only during the execution of the function and bears no relation to any object referred to by the same name at other times. Any number of variables can be made local to a function by appending each (preceded by a semicolon) to the function header. Compare the following behavior of the function $SUM3$, which has a local variable I , with the behavior of the function $SUM2$ in which I is global:

$\nabla SUM3; I$	$\nabla SUM2$
[1] $S \leftarrow 0$	[1] $S \leftarrow 0$
[2] $I \leftarrow 0$	[2] $I \leftarrow 0$
[3] $S \leftarrow S + I$	[3] $S \leftarrow S + I$
[4] $I \leftarrow I + 1$	[4] $I \leftarrow I + 1$
[5] $\rightarrow 3 \times 1 I \leq N$	[5] $\rightarrow 3 \times 1 I \leq N$
[6] ∇	[6] ∇
$I \leftarrow 20$	$I \leftarrow 20$
$N \leftarrow 5$	$N \leftarrow 5$
$SUM3$	$SUM2$
S	S
15	15
I	I
20	6

Since I is local to the function $SUM3$, execution of $SUM3$ has no effect on the variable I referred to before and after the use of $SUM3$.

However, if the variable K is local to a function F then any function G used within F may refer to the same variable K , unless the name K is further localized by being made local to G . For further treatment of this matter, see the section on Homonyms.

Explicit argument. A function definition of the form

```
    ∇SPH X
[1]  SUR←4×3.14159×X×X
[2]  ∇
```

defines *SPH* as a function with an explicit argument; whenever such a function is used it must be provided with an argument. For example:

```
    SPH 2
    SUR
50.26544
    SPH 1
    SUR
12.56636
```

Any explicit argument of a function is automatically made local to the function; if *E* is any expression, then the effect of *SPH E* is to assign to the local variable *X* the value of the expression *E* and then execute the body of the function *SPH*. Except for having a value assigned initially, the argument variable is treated as any other local variable and, in particular, may be respecified within the function.

Explicit result. Each of the primitive functions produces a result and may therefore appear within compound expressions. For example, the expression *÷Z* produces an explicit result and may therefore appear in a compound expression such as *X÷Z*. A function definition of the form

```
    ∇Z←SP X
[1]  Z←4×3.14159×X×X
[2]  ∇
```

defines *SP* as a function with an explicit result; the variable *Z* is local, and the value it assumes at the completion of execution of the body of the function is the explicit result of the function. For example:

```
    Q←3×SP 1
    Q
37.69908
    R←2
    (SP R)×R÷3
33.51029333
```

The forms of defined functions. Functions may be defined with 2, 1, or 0 explicit arguments and either with or without an explicit result. The form of header used to define each of these six types is shown in Table 3.3. Each of the six forms permits the appending of semicolons and names to introduce local variables. The names appearing in any one header must all be distinct; e.g., the header $Z \leftarrow F Z$ is invalid.

Number of Arguments	Number of Results	
	0	1
0	∇F	$\nabla Z \leftarrow F$
1	$\nabla F Y$	$\nabla Z \leftarrow F Y$
2	$\nabla X F Y$	$\nabla Z \leftarrow X F Y$

Table 3.3: FORMS OF DEFINED FUNCTIONS

It is not obligatory either for the arguments of a defined function to be used within the body, or for the result variable to be specified. A function definition which does not assign a value to the result variable will engender a value error report when it is used within a compound expression. This behavior permits a function to be defined with a restricted domain, by testing the argument(s) and branching out in certain cases without specifying a result. For example:

```

       $\nabla Z \leftarrow SQRT X$ 
[1]    $\rightarrow 0 \wedge X < 0$ 
[2]    $Z \leftarrow X * .5 \nabla$ 
       $Q \leftarrow SQRT 16$ 
       $Q$ 
4
       $Q \leftarrow SQRT \bar{16}$ 
VALUE ERROR
       $Q \leftarrow SQRT \bar{16}$ 
      ^

```

Use of defined functions. A defined function may be used in the same ways that a primitive function may. In particular, it may be used within the definition of another function. For example, the function *HYP* determines the hypotenuse of a right triangle of sides *A* and *B* by using the square root function *SQRT*:

```

      ∇Z←SQRT X
[1]   Z←X*.5∇

      ∇H←A HYP B
[1]   H←SQRT (A*2)+B*2∇

      5 HYP 12
13

```

A defined function must be used with the same number of arguments as appear in its header.

Recursive function definition. A function *F* may be used in the body of its own definition, in which case the function is said to be recursively defined. The following program *FAC* shows a recursive definition of the factorial function. The heart of the definition is statement 2, which determines factorial *N* as the product of *N* and *FAC N-1*, except for the case *N=0* when it is determined (by statement 4) as 1:

```

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

```

Trace control. A trace is an automatic type-out of information generated by the execution of a function as it progresses. In a complete trace of a function *P*, the number of each statement executed is typed out in brackets, preceded by the function name *P* and followed by the final value produced by the statement. The trace is useful in analyzing the behavior of a defined function, particularly during its design.

The tracing of *P* is controlled by the trace vector for *P*, denoted by *TΔP*. If one types *TΔP←2 3 5* then statements 2,3,and 5 will be traced in any subsequent execution of *P*. More generally, the value assigned to the trace vector may be any vector of integers. Typing *TΔP←0* will discontinue tracing of *P*. A complete trace of *P* is set up by entering *TΔP←1N*, where *N* is the number of statements in *P*.

MECHANICS OF FUNCTION DEFINITION

When a function definition is opened (by typing a ∇ followed by a header), the system automatically types successive statement numbers enclosed in brackets and accepts successive entries as the statements forming the body of the definition. The system is therefore said to be in definition mode, as opposed to the execution mode which prevails outside of function definition.

There are several devices which may be used during function definition to revise and display the function being defined. After function definition has been closed, there are convenient ways to re-open the definition so that these same devices may be used for further revision or display.

Labels. If a statement occurring in the body of a function definition is prefaced by a name and a colon, then at the end of the definition the name is assigned a value equal to the statement number. A variable specified in this way is called a label. Labels are used to advantage in branches when it is expected that a function definition may be changed for one reason or another, since a label automatically assumes the new value of the statement number of its associated statement as statements are inserted or deleted.

Revision. Any statement number (including one typed by the system) can be overridden by typing [N], where N is any positive number less than 10000, with or without a decimal point and with at most four digits to the right of the decimal point. If N is zero, it refers to the header line of the function.

If any statement number is repeated, the statement following it supersedes the earlier specification of the statement. If any statement is empty -- that is, the bracketed statement number was immediately followed by both a linefeed and a carriage return (a carriage return alone is vacuous) -- it is deleted.

When the function definition mode is ended, the statements are reordered according to their statement numbers and the statement numbers are replaced by the integers 1,2,3, and so on. Labels are assigned appropriate values.

The particular statement on which the closing ∇ appears is not significant, since it marks only the end of the definition mode, not necessarily the last line of the function. Moreover, the closing ∇ may be entered either alone or at the end of a statement.

Display. During function definition, statement N can be displayed by overriding the line number with [N \square]. After the display, the system awaits replacement of statement N. Typing [\square] displays the entire function, including the header and the opening and closing ∇ , and awaits entry of the next statement; typing [\square N] displays all statements from N onward and awaits replacement of the last statement. Executing an attention signal will stop any display.

Line editing. During function definition, statement N can be modified by the following mechanism:

1. Type [N \square M] where M is an integer.
2. Statement N is automatically displayed and the carriage stops under position M.
3. A letter or decimal digit or the symbol / may be typed under any of the positions in the displayed statement. Any other characters typed in this mode are ignored. The ordinary rules for backspace and linefeed apply.
4. When the carriage is returned, statement N is re-displayed. Each character understruck by a / is deleted, each character understruck by a digit K is preceded by K added spaces, and each character understruck by a letter is preceded by $5 \times R$ spaces, where R is the rank of the letter in the alphabet. Finally, the carriage moves to the first injected space and awaits the typing of modifications to the statement in the usual manner. The final effect is to define the statement exactly as if the entry had been made entirely from the keyboard; in particular, a completely blank sequence leaves the statement unchanged.

If the statement number itself is changed during the editing procedure, the statement affected is determined by the new statement number; hence statement N remains unchanged. This permits statements to be moved, with or without modification.

Reopening function definition. If a function R is already defined, the definition mode for that function can be re-established by entering ∇R alone; the rest of the function header must not be entered. The system responds by typing $[N+1]$, where N is the number of statements in R . Function definition then proceeds in the normal manner.

Function definition may also be established with editing or display requested on the same line. For example, $\nabla R[3]X+X+1$ initiates editing by entering a new line 3 immediately. The system responds by typing $[4]$ and awaiting continuation. The entire process may be accomplished on a single line. Thus, $\nabla R[3]X+X+1\nabla$ opens the definition of R , enters a new line 3, and terminates the definition mode. Also, $\nabla R[\]\nabla$ causes the entire definition of R to be displayed, after which the system returns to execution mode.

Similar expressions involving display are also permissible, for example, $\nabla R[\]3\nabla$ or $\nabla R[\]$ or $\nabla R[2\]10$.

Locked functions. If the symbol ∇ (formed by a ∇ overstruck with a \sim and called del-tilde) is used instead of ∇ to open or close a function definition, the function becomes locked. A locked function cannot be revised or displayed in any way. Moreover, an error stop within the function will print only the function name and statement number, not the statement. Finally, the associated stop control (see next section) and trace control vectors cannot be changed after the function is locked.

Locked functions are used to keep a function definition proprietary. For example, in an exercise in which a student is required to determine the behavior of a function by using it for a variety of arguments, locking a function prevents him from displaying its definition.

Deletion of functions and variables. A function F (whether locked or not) is deleted by the command $)ERASE F$ (see Table 2.1). It may also be deleted by deleting every one of its statements. A variable may be deleted only by the erase command.

System command entered during function definition. A system command entered during function definition will not be accepted as a statement in the definition. Some commands, such as $)COPY$, will be rejected with the message $NOT WITH OPEN DEFINITION$ (see Table 2.1); most will be executed immediately.

SUSPENDED FUNCTION EXECUTION

Suspension. The execution of a function F may be stopped before completion in a variety of ways: by an error report, by an attention signal, or by the stop control vector $S\Delta F$ treated below. In any case, the function is still active and its execution can later be resumed. In this state the function is said to be suspended. Typing $\rightarrow K$ will cause execution of the suspended function to be resumed, beginning with statement K .

Whatever the reason for suspension, the statement or statement number displayed is the next one to have been executed. A branch to that statement number will cause normal continuation of the function execution, and a branch out ($\rightarrow 0$) will terminate execution of the function.

The function $r26$ (described in the section on System Dependent Functions) yields the number of the statement next to be executed. Hence the expression $\rightarrow r26$ provides a safe and convenient way to cause normal resumption of execution.

In the suspended state all normal activities are possible. In particular, the system is in a condition to:

1. execute statements or system commands.
2. resume execution of the function at an arbitrary point N (by entering $\rightarrow N$).
3. reopen the definition of any function which is not pendent. The term pendent is defined in the discussion of the state indicator below.

If function execution is interrupted by a disconnect, the function is suspended and the resulting active workspace is automatically saved under the name *CONTINUE*, as noted in Part 2.

State indicator. Typing $)SI$ causes a display of the state indicator; a typical display has the following form:

```
)SI
H[7] *
G[2]
F[3]
```

The foregoing display indicates that execution was halted just before executing statement 7 of the function H ,

that the current use of function H was invoked in statement 2 of function G , and that the use of function G was in turn invoked in statement 3 of F . The * appearing to the right of $H[7]$ indicates that the function H is suspended; the functions G and F are said to be pendent.

Further functions can be invoked when in the suspended state. Thus if G were now invoked and a further suspension occurred in statement 5 of Q , itself invoked in statement 8 of G , a subsequent display of the state indicator would appear as follows:

```

        )SI
Q[5]  *
G[8]
H[7]  *
G[2]
F[3]
```

The entire sequence from the last to the preceding suspension can be cleared by typing a branch with no argument (that is, \rightarrow). This behavior is illustrated by continuing the foregoing example as follows:

```

        →
        )SI
H[7]  *
G[2]
F[3]
```

Repeated use of \rightarrow will clear the state indicator completely. The cleared state indicator displays as a blank line.

Stop Control. The stop vector for a function P is denoted by $S\Delta P$. It is set in the same manner as the trace vector (i.e., by $S\Delta P\leftarrow I$, where the vector I specifies the numbers of the statements controlled), and stops execution just before each of the specified statements. At each stop, the function name and the line number of the statement next to be executed are printed. After the stop the system is in the normal suspended state; resumption of execution may therefore be initiated by a branch.

Trace control and stop control can be used in conjunction. Moreover, either of the control vectors may be set within functions. In particular, they may be set by expressions which initiate tracing or halts only for certain values of certain variables.

HOMONYMS

Variable names. The use of local variables introduces the possibility of having more than one object in a workspace with the same name. Confusion is avoided by the following rule: when a function is executed, its local variables supersede, for the duration of the execution, other objects of the same name. A name may, therefore, be said to have one active referent and (possibly) several latent referents.

The complete set of referents of a name can be determined with the aid of the SIV list (state indicator with local variables), whose display is initiated by the command `)SIV`. The SIV list contains the information provided by the command `)SI`, augmented by the names of the variables local to each function. A sample display follows:

```

)SIV
G[7] * Z X I
F[4]   P J
Q[3] * C X T
R[2]   P
G[3]   Z X I

```

If the SIV list is scanned downward, from the top, the first occurrence of a variable is the point at which its active referent was introduced; lower occurrences are the points at which currently latent referents were introduced; and if the name is not found at all, its referent is global, and should be sought for with the commands `)FNS`, `)VARS`, or `)GRPS`.

As the state indicator is cleared (by `→`, or by the continuation to completion of halted functions), latent referents become active in the sequence summarized, for the preceding SIV list, by the following diagram:

	Z	X	I	P	J	C	T	A	B
G	↓	↓	↓						
F				↓	↓				
Q		↓				↓	↓		
R				↓					
G	↓	↓	↓						
Global	↓	↓	↓	↓	↓	↓	↓	↓	↓

The currently active referent of a name holds down to and including the execution of the function listed at the point of the first arrow, because of localization of the

name within that function. The first latent referent becomes active when that function is completed, and holds down to the next arrow; and so forth until the state indicator is completely cleared, at which point there are no longer any latent referents, and all active referents are global objects.

Function names. All function names are global. In the foregoing example, therefore, a function named *P* cannot be used within the function *R* or within any of the functions employed by *R*, since the local variable name *P* makes the function *P* inaccessible. However, even in such circumstances, the opening of function definition for such a function *P* is possible. (Moreover, as stated in Part 2, system commands concern global objects only, regardless of the current environment.)

This scheme of homonyms is easy to use and relatively free from pitfalls. It can, however, lead to seeming anomalies as indicated by the following example (shown to the authors by J.C.Shaw) of two pairs of functions which differ only in the name used for the argument:

<pre> [1] ∇Z←F X Z←X+Y∇ </pre>	<pre> [1] ∇Z←F X Z←X+Y∇ </pre>
<pre> [1] ∇Z←G Y Z←F Y∇ Y←3 G 4 </pre>	<pre> [1] ∇Z←G R Z←F R∇ Y←3 G 4 </pre>
8	7

INPUT AND OUTPUT

The following function determines the value of an amount *A* invested at interest *B*[1] for a period of *B*[2] years:

```

∇Z←A CPI B
[1]  Z←A×(1+.01×B[1])*B[2]∇

```

For example:

```

1000 CPI 5 4
1215.50625

```

The casual user of such a function might, however, find it onerous to remember the positions of the various arguments and whether the interest rate is to be entered as the actual rate (e.g., .05) or in percent (e.g., 5). An exchange of the following form might be more palatable:

```

      CI
ENTER CAPITAL AMOUNT IN DOLLARS
□:
      1000
ENTER INTEREST IN PERCENT
□:
      5
ENTER PERIOD IN YEARS
□:
      4
RESULT IS 1215.50625

```

It is necessary that each of the keyboard entries (1000, 5, and 4) occurring in such an exchange be accepted not as an ordinary entry (which would only evoke the response 1000, etc.), but as data to be used within the function *CI*. Facilities for this are provided in two ways, termed evaluated input, and character input.

The definition of the function *CI* is shown at the end of this section.

Evaluated input. The quad symbol □ appearing anywhere other than immediately to the left of a specification arrow accepts keyboard input as follows: the two symbols □: are printed, the paper is spaced up one line, and the keyboard unlocks. Any valid expression entered at this point is evaluated and the result is substituted for the quad. For example:

```

      ∇Z←F
[1]   Z←4×□*2
[2]   ∇
      F
□:
      3
36
      F
□:
      3÷2
9

```


An invalid entry in response to request for a quad input induces an appropriate error report, after which input is again awaited at the same point. A system command entered will be executed, after which (except in the case of one which replaces the active workspace) a valid expression will again be awaited. An empty input (i.e., a carriage return alone or spaces and a carriage return) is rejected and the system again prints the symbols □: and awaits input.

The symbols □: are printed to alert the user to the type of input expected; they can be changed by the library function *SFEI* as described in Part 4.

Character input, The quote-quad symbol □ (i.e., a quad overstruck with a quote) accepts character input: the keyboard unlocks at the left margin and data entered are accepted as characters. For example:

```

      X←□
CAN'T          (Quote-quad input, not indented)
      X
CAN'T

```

Escape from input loop. If evaluated or character input occurs within an endless loop in a function, it may be impossible to escape by the usual device of striking the attention button. Escape from □ input can be achieved by entering →. Escape from □ input can be achieved by typing the three letters *OUT*, in that order, but with a backspace between each pair so that they all overstrike. The effect is exactly as if the symbol → were entered while suspended.

Normal output. The quad symbol appearing immediately to the left of a specification arrow indicates that the value of the expression to the right of the arrow is to be printed. Hence, □←*X* is equivalent to the statement *X*. The longer form □←*X* is useful when employing multiple specification. For example, □←*Q*←*X**2 assigns to *Q* the value *X**2 and then prints the value of *X**2.

The page width (measured in characters) may be set to any value *N* in the range 30-130 by entering the command *)WIDTH N*. It may also be set by the library function *WIDTH* which may be used within a defined function. (See Part 4.)

Heterogeneous output. A sequence of expressions separated by semi-colons will cause the values of the expressions to be printed, with no intervening carriage returns or spaces except those implicit in the display of the values.

The primary use of this form is for output in which some of the expressions yield numbers and some yield characters. For example, if $X \leftarrow 2.14$, then:

```
'THE VALUE OF X IS ';X
THE VALUE OF X IS 2.14
```

A further example of mixed output is furnished by the definition of the function *CI* which introduced the present section:

```
∇CI;A;I;Y
[1] 'ENTER CAPITAL AMOUNT IN DOLLARS'
[2] A←□
[3] 'ENTER INTEREST IN PERCENT'
[4] I←□
[5] 'ENTER PERIOD IN YEARS'
[6] Y←□
[7] 'RESULT IS ';A*(1+.01*I)*Y∇
```

RECTANGULAR ARRAYS

Introduction. A single element of a rectangular array can be selected by specifying its indices; the number of indices required is called the dimensionality or rank of the array. Thus a vector is of rank 1, a matrix (in which the first index selects a row and the second a column) is of rank 2, and a scalar (since it permits no selection by indices) is an array of rank 0. Rectangular arrays of higher rank may be used, and are called 3-dimensional, 4-dimensional, etc.

This section treats the reshaping and indexing of arrays, and the form of array output. The following section treats the four ways in which the basic scalar functions are extended to arrays, and the next section thereafter treats the definition of certain mixed functions on arrays.

Vectors, dimension, catenation. If X is a vector, then ρX denotes its dimension. For example, if $X \leftarrow 2\ 3\ 5\ 7\ 11$, then ρX is 5, and if $Y \leftarrow 'ABC'$, then ρY is 3. A single character entered in quotes or in response to a \square input is a scalar, not a vector of dimension 1; this parallels the case of a single number, which is also a scalar.

Catenation chains two vectors (or scalars) together to form a vector; it is denoted by a comma. For example:

```

      X ← 2 3 5 7 11
      X, X
2 3 5 7 11 2 3 5 7 11

```

In general, the dimension of X, Y is equal to the total number of elements in X and Y . A numeric vector cannot be catenated with a character vector. (However, see Heterogeneous Output.)

Matrices, dimension, ravel. The monadic function ρ applied to an array A yields the size of A , that is, a vector whose components are the dimensions of A . For example, if A is the matrix

```

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

```

of three rows and four columns, then ρA is the vector 3 4.

Since ρA contains one component for each coordinate of A , the expression $\rho \rho A$ is the rank of A . Table 3.4 illustrates the values of ρA and $\rho \rho A$ for arrays of rank 0 (scalars) up to rank 3. In particular, the function ρ applied to a scalar yields an empty vector.

Type of Array	ρA	$\rho \rho A$	$\rho \rho \rho A$
Scalar		0	1
Vector	N	1	1
Matrix	$M\ N$	2	1
3-Dimensional	$L\ M\ N$	3	1

Table 3.4: DIMENSION AND RANK VECTORS

The monadic function ravel is denoted by a comma; when applied to any array A it produces a vector whose elements are the elements of A in row order. For example, if A is the matrix

```

      2   4   6   8
     10  12  14  16
     18  20  22  24

```

and if $V \leftarrow A$ then V is a vector of dimension 12 whose elements are the integers 2 4 6 8 10 12 ... 24. If A is a vector, then $,A$ is equivalent to A ; if A is a scalar, then $,A$ is a vector of dimension 1.

Reshape. The dyadic function ρ reshapes its right argument to the dimension specified by its left argument. If $M \leftarrow D \rho V$, then M is an array of dimension D whose elements are the elements of V . For example, $2\ 3 \rho 1\ 2\ 3\ 4\ 5\ 6$ is the matrix

```

      1  2  3
      4  5  6

```

If N , the total number of elements required in the array $D \rho V$, is equal to the dimension of the vector V , then the ravel of $D \rho V$ is equal to V . If N is less than ρV , then only the first N elements of V are used; if N is greater than ρV , then the elements of V are repeated cyclically. For example, $2\ 3 \rho 1\ 2$ is the matrix

```

      1  2  1
      2  1  2

```

and $3\ 3 \rho 1\ 0\ 0\ 0$ is the identity matrix

```

      1  0  0
      0  1  0
      0  0  1

```

More generally, if A is any array, then $D_{\rho}A$ is equivalent to D_{ρ},A . For example, if A is the matrix

```

1  2  3
4  5  6

```

then ${}^3_5\rho A$ is the matrix

```

1  2  3  4  5
6  1  2  3  4
5  6  1  2  3

```

The expressions ${}^0\rho X$ and ${}^0_3\rho X$ and ${}^3_0\rho X$ and ${}^0_0\rho X$ are all valid; any one or more of the dimensions of an array may be zero.

Uses of empty arrays. A vector of dimension zero contains no components and is called an empty vector. Three expressions which yield empty vectors are 10 and $'$ and ρ applied to any scalar. An empty vector prints as a blank line.

One important use of the empty vector has already been illustrated: when one occurs as the argument of a branch, the effect is to continue the normal sequence.

The following function for determining the representation of any positive integer N in a base B number system shows a typical use of the empty vector in initializing a vector Z which is to be built up by successive catenations:

```

      VZ←B BASE N
[1]   Z← ${}^10$ 
[2]   Z←(B|N),Z
[3]   N←[N÷B
[4]   →2×N>0∇
      10 BASE 1776
1  7  7  6
      8 BASE 1776
3  3  6  0

```

Empty arrays of higher rank can be useful in analogous ways in conjunction with the expansion function described in the section on Mixed Functions.

Indexing. If X is a vector and I is a scalar, then $X[I]$ denotes the I th element of X . For example, if $X \leftarrow 2\ 3\ 5\ 7\ 11$ then $X[2]$ is 3.

If the index I is a vector, then $X[I]$ is the vector obtained by selecting from X the elements indicated by successive components of I . For example, $X[1\ 3\ 5]$ is 2 5 11 and $X[5\ 4\ 3\ 2\ 1]$ is 11 7 5 3 2 and $X[13]$ is 2 3 5. If the elements of I do not belong to the set of indices of X , then the expression $X[I]$ induces an index error report.

In general, $\rho X[I]$ is equal to ρI . In particular, if I is a scalar, then $X[I]$ is a scalar, and if I is a matrix, then $X[I]$ is a matrix. For example:

```
A ← 'ABCDEFG'
M ← 4 3 1 4 2 1 4 4 1 2 4 1 4
M
3 1 4
2 1 4
4 1 2
4 1 4
A[M]
```

```
CAD
BAD
DAB
DAD
```

If M is a matrix, then M is indexed by a two-part list of the form $I;J$ where I selects the row (or rows) and J selects the column (or columns). For example, if M is the matrix

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

then $M[2;3]$ is the element 7 and $M[1\ 3; 2\ 3\ 4]$ is the matrix

```
2 3 4
10 11 12
```

In general, $\rho M[I;J]$ is equal to $(\rho I), \rho J$. Hence if I and J are both vectors, then $M[I;J]$ is a matrix; if both I and J are scalars, $M[I;J]$ is a scalar; if I is a vector and J is a scalar (or vice versa), $M[I;J]$ is a vector. The indices are not limited to vectors, but may be of higher rank. For example, if I is a 3 by 4 matrix, and J is a vector of dimension 6, then $M[I;J]$ is of dimension 3 4 6, and $M[J;I]$ is of dimension 6 3 4. In particular, if T and P and Q are matrices, and if $R \leftarrow T[P;Q]$, then R is an array of rank 4 and $R[I;J;K;L]$ is equal to $T[P[I;J];Q[K;L]]$.

The form $M[I;]$ indicates that all columns are selected, and the form $M[;J]$ indicates that all rows are selected. For example, $M[2;]$ is 5 6 7 8 and $M[;2]$ is

```

      2  1
      6  5
      10 9

```

The following example illustrates the use of a matrix indexing a matrix to obtain a three-dimensional array:

```

M ← 4 3 ρ 3 1 4 2 1 4 4 1 2 4 1 4
M
3  1  4
2  1  4
4  1  2
4  1  4
  M[M;]
4  1  2
3  1  4
4  1  4
2  1  4
3  1  4
4  1  4
4  1  4
3  1  4
2  1  4
4  1  4
3  1  4
4  1  4

```

Permutations are an interesting use of indexing. A vector P whose elements are some permutation of its own indices is called a permutation of order ρP . For example, 3 1 4 2 is a permutation of order 4. If X is any vector of the same dimension as P , then $X[P]$ produces a permutation of X . Moreover, if ρP is equal to $(\rho M)[1]$, then $M[P;]$ permutes the column vectors of M (i.e., interchanges the rows of M) and is called a column permutation. Similarly, if ρP equals $(\rho M)[2]$, then $M[;P]$ is a row permutation of M .

Indexing on the left. An array appearing to the left of a specification arrow may be indexed, in which case only the selected positions are affected by the specification. For example:

```

      X+2 3 5 7 11
      X[1 3]+6 8
      X
6 3 8 7 11

```

The normal restrictions on indexing apply; in particular, a variable which has not already been assigned a value cannot be indexed, and an out-of-range index value cannot be used.

Index origin. In l-origin indexing, $X[1]$ is the leading element of the vector X and $X[\rho X]$ is the last element. In 0-origin indexing, $X[0]$ is the leading element and $X[-1+\rho X]$ is the last. 0-origin indexing is instituted by the command `)ORIGIN 0`. The command `)ORIGIN 1` restores l-origin indexing. The index origin in effect applies to all coordinates of all rectangular arrays.

The function `ORIGIN` in Library 1 `WSFNS` may also be used to control the index origin. It may be executed within a function. (See Part 4.)

In certain expressions such as $+/[J]M$ and $K\phi[J]M$ (to be treated more fully in the two following sections), the value of J determines the coordinate of the array M along which the function is to be applied. Since the numbering of coordinates follows the index origin, a change of index origin also affects the behavior of such expressions.

The index origin also affects four other functions, the monadic and the dyadic forms of `?` and `ι`. The expression ιN yields a vector of the first N integers beginning with the index origin. Hence $X[\iota N]$ selects the first N components of X in either origin. Moreover, $\iota 1$ is a one-element vector having the value 0 in 0-origin and 1 in l-origin; $\iota 0$ is an empty vector in either origin.

The index origin remains associated with a workspace; in particular, the index origin of an active workspace is not affected by a copy command. A clean workspace provided on sign-on or by the command)CLEAR is in l-origin. All definitions and examples in this text are expressed in l-origin.

Array output. Character arrays print with no spaces between components in each row; other arrays print with at least one space. If a vector or a row of a matrix requires more than one line, succeeding lines are indented.

A matrix prints with all columns aligned and with a blank line before the first row. A matrix of dimension $N,1$ prints as a single column.

FUNCTIONS ON ARRAYS

There are four ways in which the scalar functions of Table 3.2 extend to arrays: element-by-element, reduction, inner product, and outer product. Reduction and outer product are defined on any arrays, but the other two extensions are defined only on arrays whose sizes satisfy a certain relationship called conformability. For the element-by-element extension, conformability requires that the shapes of the arrays agree, unless one is a scalar. The requirements for inner product are shown in Table 3.6.

Scalar functions. All of the scalar functions of Table 3.2 are extended to arrays element by element. Thus if M and N are matrices of the same size, f is a scalar function, and $P \leftarrow MfN$, then $P[I;J]$ equals $M[I;J]fN[I;J]$, and if $Q \leftarrow fN$, then $Q[I;J]$ is equal to $fN[I;J]$.

If M and N are not of the same size, then MfN is undefined (and induces a length or rank error report) unless one or other of M and N is a scalar or one-element array, in which case the single element is applied to each element of the other argument. In particular, a scalar versus an empty array produces an empty array.

An expression or function definition which employs only scalar functions and scalar constants extends to arrays like a scalar function.

Reduction. The sum-reduction of a vector X is denoted by $+X$ and defined as the sum of all components of X . More generally, for any scalar dyadic function f , the expression f/X is equivalent to $X[1]fX[2]f\dots fX[\rho X]$, where evaluation is from rightmost to leftmost as usual. A user-defined function cannot be used in reduction.

If X is a vector of dimension zero, then f/X yields the identity element of the function f (listed in Table 3.5) if it exists; if X is a scalar or a vector of dimension 1, then f/X yields the value of the single element of X .

The result of reducing any vector or scalar is a scalar.

Dyadic Function		Identity Element	Left-Right
Times	\times	1	L R
Plus	$+$	0	L R
Divide	\div	1	R
Minus	$-$	0	R
Power	$*$	1	R
Logarithm	\otimes		None
Maximum	\lceil	$7.237\dots E75$	L R
Minimum	\lfloor	$7.237\dots E75$	L R
Residue	$ $	0	L
Circle	\circ		None
Out of	$!$	1	L
Or	\vee	0	L R
And	\wedge	1	L R
Nor	∇		None
Nand	∇^*		None
Equal	$=$	1	Apply for logical arguments only
Not equal	\neq	0	
Greater	$>$	0	
Not less	\geq	1	
Less	$<$	0	
Not greater	\leq	1	

Table 3.5: IDENTITY ELEMENTS OF PRIMITIVE SCALAR DYADIC FUNCTIONS

For a matrix M , reduction can proceed along the first coordinate (denoted by $f/[1]M$) or along the second coordinate ($f/[2]M$). The result in either case is a vector; in general, reduction applied to any non-scalar array A produces a result of rank one less than the rank of A (hence the term reduction). The numbering of coordinates follows the index origin, and an attempt to reduce along a non-existent coordinate will result in an index error.

Since $+/[1]M$ scans over the row index of M it sums each column vector of M , and $+/[2]M$ sums the row vectors of M . For example, if M is the matrix

$$\begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array}$$

then $+/[1]M$ is 5 7 9 and $+/[2]M$ is 6 15.

In reducing along the last coordinate of an array, the coordinate indicator may be elided -- thus $+/M$ denotes summing over each of the rows of M and $+/V$ denotes summing over the last (and only) coordinate of the vector V .

Reduction over the first coordinate of M by a function f may be obtained by using the expression f/M . The symbol $/$ is formed by overstriking the solidus with the minus sign.

Inner product. The familiar matrix product is denoted by $C \leftarrow A \cdot \times B$. If A and B are matrices, then C is a matrix such that $C[I;J]$ is equal to $+/A[I;] \times B[;J]$. A similar definition applies to $A f \cdot g B$ where f and g are any of the standard scalar dyadic functions.

If A is a vector and B is a matrix, then C is a vector such that $C[J]$ is equal to $+/A \times B[;J]$. If B is a vector and A is a matrix, then C is a vector such that $C[I]$ is equal to $+/A[I;] \times B$. If both A and B are vectors, then $A \cdot \times B$ is the scalar $+/A \times B$.

The last dimension of the pre-multiplier A must equal the first dimension of the post-multiplier B , except that if either argument is a scalar, it is extended in the usual way. For non-scalar arguments, the dimension of the result is equal to $(\bar{1} \uparrow \rho A), 1 \uparrow \rho B$. (See the function drop in the section on Mixed Functions.) In other words, the dimension of the result is equal to $(\rho A), \rho B$ except for the two inner dimensions $(\bar{1} \uparrow \rho A$ and $1 \uparrow \rho B)$, which must agree and which are eliminated by the reduction over them.

Definitions for various cases are shown in Table 3.6.

Outer product. The outer product of two arrays X and Y with respect to a standard scalar dyadic function g is denoted by $X \cdot \cdot g Y$ and yields an array of dimension $(\rho X), \rho Y$, formed by applying g to every pair of components of X and Y .

ρA	ρB	$\rho A f. g B$	Conformability requirements	Definition $Z \leftarrow A f. g B$
	V			$Z \leftarrow f / A g B$
U	V		$U = V$	$Z \leftarrow f / A g B$
U	V			$Z \leftarrow f / A g B$
T	V	W		$Z[I] \leftarrow f / A g B[; I]$
T	U	T		$Z[I] \leftarrow f / A[I;] g B$
T	U	V	$U = V$	$Z[I] \leftarrow f / A g B[; I]$
T	U	V	$U = V$	$Z[I] \leftarrow f / A[I;] g B$
T	U	V	$U = V$	$Z[I; J] \leftarrow f / A[I;] g B[; J]$

Table 3.6: INNER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTIONS f AND g

If X and Y are vectors and $Z \leftarrow X \circ. g Y$, then $Z[I; J]$ is equal to $X[I] g Y[J]$. For example:

$X \leftarrow 13$
 $Y \leftarrow 14$
 $X \circ. \times Y$

1 2 3 4
2 4 6 8
3 6 9 12

$X \circ. \geq Y$

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

If X is a vector and Y is a matrix, and $Z \leftarrow X \circ. g Y$, then $Z[I; J; K]$ is equal to $X[I] g Y[J; K]$. Definitions for various cases are shown in Table 3.7.

ρA	ρB	$\rho A \circ. g B$	Definition $Z \leftarrow A \circ. g B$
			$Z \leftarrow A g B$
U	V	V	$Z[I] \leftarrow A g B[I]$
U	V	U	$Z[I] \leftarrow A[I] g B$
U	V	U	$Z[I; J] \leftarrow A[I] g B[J]$
T	V	V	$Z[I; J] \leftarrow A g B[I; J]$
T	U	T	$Z[I; J] \leftarrow A[I; J] g B$
T	U	U	$Z[I; J; K] \leftarrow A[I] g B[J; K]$
T	U	T	$Z[I; J; K] \leftarrow A[I; J] g B[K]$
T	U	T	$Z[I; J; K; L] \leftarrow A[I; J] g B[K; L]$

Table 3.7: OUTER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTION g

MIXED FUNCTIONS

Introduction. The scalar functions listed in Table 3.2 each take a scalar argument (or arguments) and yield a scalar result; each is also extended element by element to arrays. The mixed functions of Table 3.8, on the other hand, may be defined on vector arguments to yield a scalar result or a vector result, or may be defined on scalar arguments to yield a vector result. In extending these definitions to arrays of higher rank, it may therefore be necessary to specify which coordinate of an array the mixed function is applied to. The expression $[J]$ following a function symbol indicates that the function is applied to the J th coordinate. If the expression is elided, the function applies to the last coordinate of the argument array. These conventions agree with those used earlier in reduction.

The numbering of coordinates follows the index origin.

Transpose. The expression ${}^2 1\phi M$ yields the transpose of the matrix M ; that is, if $R \leftarrow {}^2 1\phi M$, then each element $R[I;J]$ is equal to $M[J;I]$. For example:

		M						${}^2 1\phi M$		
	1	2	3	4		1	5	9		
	5	6	7	8		2	6	10		
	9	10	11	12		3	7	11		
						4	8	12		

If P is any permutation of order $\rho \rho A$, then $P\phi A$ is an array similar to A except that the coordinates are permuted: the I th coordinate becomes the $P[I]$ th coordinate of the result. Hence, if $R \leftarrow P\phi A$, then $(\rho R)[P]$ is equal to ρA . For example:

		$A \leftarrow {}^2 3 5 7 \rho 1 2 1 0$				
		ρA				
2	3	5	7			
		$P \leftarrow {}^2 3 4 1$				
		$\rho P\phi A$				
7	2	3	5			

Name	Sign ¹	Definition or example ²
Size	ρA	$\rho P \leftrightarrow 4$ $\rho E \leftrightarrow 3\ 4$ $\rho 5 \leftrightarrow 1\ 0$
Reshape	$V\rho A$	Reshape A to dimension V $3\ 4\rho 1\ 1\ 2 \leftrightarrow E$ $1\ 2\rho E \leftrightarrow 1\ 1\ 2$ $0\rho E \leftrightarrow 1\ 0$
Ravel	$,A$	$,A \leftrightarrow (\times/\rho A)\rho A$ $,E \leftrightarrow 1\ 1\ 2$ $\rho, 5 \leftrightarrow 1$
Catenate	V, V	$P, 1\ 2 \leftrightarrow 2\ 3\ 5\ 7\ 1\ 2$ $'T', 'HIS' \leftrightarrow 'THIS'$
Index ^{3,4}	$V[A]$	$P[2] \leftrightarrow 3$ $P[4\ 3\ 2\ 1] \leftrightarrow 7\ 5\ 3\ 2$
	$M[A; A]$	$E[1\ 3; 3\ 2\ 1] \leftrightarrow 3\ 2\ 1$ $11\ 10\ 9$
	$A[A; \dots]$ $\dots; A]$	$E[1;] \leftrightarrow 1\ 2\ 3\ 4$ $ABCD$ $E[; 1] \leftrightarrow 1\ 5\ 9$ $'ABCDEFGHIJKL'[E] \leftrightarrow EFGH$ $IJKL$
Index generator ³	$1S$	First S integers $14 \leftrightarrow 1\ 2\ 3\ 4$ $10 \leftrightarrow$ an empty vector
Index of ³	$V1A$	Least index of A in V , or $1+\rho V$ $P13 \leftrightarrow 2$ $5\ 1\ 2\ 5$ $P1E \leftrightarrow 3\ 5\ 4\ 5$ $4\ 414 \leftrightarrow 1$ $5\ 5\ 5\ 5$
Take	$V \uparrow A$	Take or drop $ V[I] $ first elements of coordinate I $2\ 3 \uparrow X \leftrightarrow ABC$ $(V[I] \geq 0)$ or last ($V[I] < 0$) elements of coordinate I EFG
Drop	$V \downarrow A$	
Grade up ^{3,5}	ΔA	The permutation which would order A (ascending or descending) $\Delta 3\ 5\ 3\ 2 \leftrightarrow 4\ 1\ 3\ 2$ $\nabla 3\ 5\ 3\ 2 \leftrightarrow 2\ 1\ 3\ 4$
Grade down ^{3,5}	∇A	
Compress ⁵	V/A	$1\ 0\ 1\ 0/P \leftrightarrow 2\ 5$ $1\ 0\ 1\ 0/E \leftrightarrow 5\ 7$ $9\ 11$ $1\ 0\ 1/[1]E \leftrightarrow 1\ 2\ 3\ 4 \leftrightarrow 1\ 0\ 1 \uparrow E$ $9\ 10\ 11\ 12$
Expand ⁵	$V \setminus A$	$1\ 0\ 1 \setminus 1\ 2 \leftrightarrow 1\ 0\ 2$ $1\ 0\ 1\ 1\ 1 \setminus X \leftrightarrow E\ FGH$ $I\ JKL$
Reverse ⁵	ϕA	$DCBA$ $IJKL$ $\phi X \leftrightarrow HGFE$ $\phi[1]X \leftrightarrow \ominus X \leftrightarrow EFGH$ $LKJI$ $\phi P \leftrightarrow 7\ 5\ 3\ 2$ $ABCD$
Rotate ⁵	$A \phi A$	$3 \phi P \leftrightarrow 7\ 2\ 3\ 5 \leftrightarrow ^{-1} \phi P$ $1\ 0\ ^{-1} \phi X \leftrightarrow BCDA$ $EFGH$ $LIJK$
Transpose	$V \phi A$	Coordinate I of A becomes coordinate $V[I]$ of result $2\ 1 \phi X \leftrightarrow AEI$ BFJ CGK DHL
	ϕA	Transpose last two coordinates $\phi E \leftrightarrow 2\ 1 \phi E$
Membership	$A \in A$	$\rho W \in Y \leftrightarrow \rho W$ $0\ 1\ 1\ 0$ $E \in P \leftrightarrow 1\ 0\ 1\ 0$ $P \in 14 \leftrightarrow 1\ 1\ 0\ 0$ $0\ 0\ 0\ 0$
Decode	$V \downarrow V$	$10 \downarrow 1\ 7\ 7\ 6 \leftrightarrow 1776$ $24\ 60\ 60 \downarrow 1\ 2\ 3 \leftrightarrow 3723$
Encode	$V \uparrow S$	$24\ 60\ 60 \uparrow 3723 \leftrightarrow 1\ 2\ 3$ $60\ 60 \uparrow 3723 \leftrightarrow 2\ 3$
Deal ³	$S ? S$	$W ? Y \leftrightarrow$ Random deal of W elements from $1Y$

Table 3.8: PRIMITIVE MIXED FUNCTIONS (see adjacent notes)

1. Restrictions on argument ranks are indicated by: *S* for scalar, *V* for vector, *M* for matrix, *A* for Any. Except as the first argument of *S*₁*A* or *S*[*A*], a scalar may be used instead of a vector. A one-element array may replace any scalar.
2. Arrays used
in examples: *P* ↔ 2 3 5 7 *E* ↔

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

X ↔

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
<i>E</i>	<i>F</i>	<i>G</i>	<i>H</i>
<i>I</i>	<i>J</i>	<i>K</i>	<i>L</i>
3. Function depends on index origin.
4. Elision of any index selects all along that coordinate.
5. The function is applied along the last coordinate; the symbols */*, **, and *⊖* are equivalent to */*, **, and *⊕*, respectively, except that the function is applied along the first coordinate. If [*S*] appears after any of the symbols, the relevant coordinate is determined by the scalar *S*.

Notes to Table 3.8

More generally, *Q⊖A* is a valid expression if *Q* is any vector of dimension *ρρA* whose elements are chosen from (and exhaust) the elements of *1*[*/Q*]. For example, if *ρρA* is equal to 3, then 1 1 2 and 2 1 1 and 1 1 1 are suitable values for *Q* but 1 3 1 is not. Just as for the case *P⊖A* where *P* is a permutation vector, the *I*th coordinate becomes the *Q*[*I*]th coordinate of *Q⊖A*. However, in this case two or more of the coordinates of *A* may map into a single coordinate of the result, thus producing a diagonal section of *A* as illustrated below:

```

A←3 3ρ19
A
1 2 3
4 5 6
7 8 9
   1 1⊖A
1 5 9

```

Table 3.9 shows the detailed definitions of transposition for a variety of cases.

Monadic transpose. The expression ϕA yields the array A with the last two coordinates interchanged. For a vector V , matrix M , and three dimensional array T , the following relations hold:

- ϕV is equivalent to $1\phi V$ (and hence to V)
- ϕM is equivalent to $2\ 1\phi M$ (ordinary matrix transpose)
- ϕT is equivalent to $1\ 3\ 2\phi T$

Rotate. If K is a scalar or one-element vector and X is a vector, then $K\phi X$ is a cyclic rotation of X defined as follows: $K\phi X$ is equal to $X[1+(\rho X)^{-1+K+\rho X}]$. For example, if $X \leftarrow 2\ 3\ 5\ 7\ 11$, then $2\phi X$ is equal to $5\ 7\ 11\ 2\ 3$, and $^{-2}\phi X$ is equal to $7\ 11\ 2\ 3\ 5$. In 0-origin indexing, the definition for $K\phi X$ becomes $X[(\rho X)|K+\rho X]$.

If the rank of X exceeds 1, then the coordinate J along which rotation is to be performed may be specified in the form $Z \leftarrow K\phi[J]X$. Moreover, the dimension of K must equal the remaining dimensions of X , and each vector along the J th coordinate of X is rotated as specified by the corresponding element of K . A scalar K is extended in the usual manner.

Case	ρR	Definition
$R \leftarrow 1\phi V$	ρV	$R \leftarrow V$
$R \leftarrow 1\ 2\phi M$	ρM	$R \leftarrow M$
$R \leftarrow 2\ 1\phi M$	$(\rho M)[2\ 1]$	$R[I;J] \leftarrow M[J;I]$
$R \leftarrow 1\ 1\phi M$	$\lfloor / \rho M$	$R[I] \leftarrow M[I;I]$
$R \leftarrow 1\ 2\ 3\phi T$	ρT	$R \leftarrow T$
$R \leftarrow 1\ 3\ 2\phi T$	$(\rho T)[1\ 3\ 2]$	$R[I;J;K] \leftarrow T[I;K;J]$
$R \leftarrow 2\ 3\ 1\phi T$	$(\rho T)[3\ 1\ 2]$	$R[I;J;K] \leftarrow T[J;K;I]$
$R \leftarrow 3\ 1\ 2\phi T$	$(\rho T)[2\ 3\ 1]$	$R[I;J;K] \leftarrow T[K;I;J]$
$R \leftarrow 1\ 1\ 2\phi T$	$(\lfloor / (\rho T)[1\ 2]), (\rho T)[3]$	$R[I;J] \leftarrow T[I;I;J]$
$R \leftarrow 1\ 2\ 1\phi T$	$(\lfloor / (\rho T)[1\ 3]), (\rho T)[2]$	$R[I;J] \leftarrow T[I;J;I]$
$R \leftarrow 2\ 1\ 1\phi T$	$(\lfloor / (\rho T)[2\ 3]), (\rho T)[1]$	$R[I;J] \leftarrow T[J;I;I]$
$R \leftarrow 1\ 1\ 1\phi T$	$\lfloor / \rho T$	$R[I] \leftarrow T[I;I;I]$

Table 3.9: TRANSPOSITION

For example, if ρX is 3 4 and J is 2, then K must be of dimension 3 and $Z[I;]$ is equal to $K[I]\phi X[I;]$. If J is 1, then ρK must be 4, and $Z[;I]$ is equal to $K[I]\phi X[;I]$. If X is a three-dimensional array, then K must be a matrix or a scalar. For example:

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

The expression $K\epsilon X$ denotes rotation along the first coordinate of X . The symbol ϵ is formed by overstriking a ϕ with a minus sign.

Reverse. If X is a vector and $R\leftarrow\phi X$, then R is equal to X except that the elements appear in reverse order. Formally, R is equal to $X[1+(\rho X)-1:\rho X]$. In 0-origin indexing, the appropriate expression is $X[-1+(\rho X)-1:\rho X]$.

If A is any array, J is a scalar or one-element vector, and $R\leftarrow\phi[J]A$, then R is an array like A except that the order of the elements is reversed along the J th coordinate. For example:

A	$\phi[1]A$	$\phi[2]A$
1 2 3	4 5 6	3 2 1
4 5 6	1 2 3	6 5 4

The expression ϕA denotes reversal along the last coordinate of A , and ϵA denotes reversal along the first coordinate. For example, if A is of rank 3, then ϕA is equivalent to $\phi[3]A$, and ϵA is equivalent to $\phi[1]A$.

Compress. The expression U/X denotes compression of X by U . If U is a logical vector (comprising elements having only the values 0 or 1) and X is a vector of the same dimension, then U/X produces a vector result of $+/U$ elements chosen from those elements of X corresponding to non-zero elements of U . For example, if $X\leftarrow 2 3 5 7 11$ and $U\leftarrow 1 0 1 1 0$ then U/X is 2 5 7 and $(\sim U)/X$ is 3 11.

To be conformable, the dimensions of the arguments must agree, except that a scalar (or one-element vector) left argument is extended to apply to all elements of the right argument. Hence $1/X$ is equal to X and $0/X$ is an empty vector. A scalar right argument is not extended. The result in every case is a vector.

If M is a matrix, then $U/[1]M$ denotes compression along the first coordinate, that is, the compression operates on each column vector and therefore deletes certain rows. It is called column compression. Similarly, $U/[2]M$ (or simply U/M) denotes row compression. The result in every case is a matrix. As in reduction, U/M denotes compression along the last coordinate, and $U\neq M$ denotes compression along the first.

Expand. Expansion is the converse of compression and is denoted by $U\backslash X$. If $Y\leftarrow U\backslash X$, then U/Y is equal to X and (if X is an array of numbers) $(\sim U)/Y$ is an array of zeros. In other words, $U\backslash X$ expands X to the format indicated by the ones in U and fills in zeros elsewhere. To be conformable, $+/U$ must equal ρX .

If X is an array of characters, then spaces are supplied rather than zeros, i.e., if $Y\leftarrow U\backslash X$ then $(\sim U)/Y$ is an array of the space character ' '. Again, $U\backslash[J]M$ denotes expansion along the J th coordinate, $U\backslash M$ denotes expansion along the last, and $U\neq M$ denotes expansion along the first. See Table 3.8 for examples of expansion.

A scalar left argument is not extended.

Decode. The expression $R\downarrow X$ denotes the value of the vector X evaluated in a number system with radices $R[1], R[2], \dots, R[\rho R]$. For example, if $R\leftarrow 24\ 60\ 60$ and $X\leftarrow 1\ 2\ 3$ is a vector of elapsed time in hours, minutes, and seconds, then $R\downarrow X$ has the value 3723, and is the corresponding elapsed time in seconds. Similarly, $10\ 10\ 10\ 10\ \downarrow\ 1\ 7\ 7\ 6$ is equal to 1776, and $2\ 2\ 2\ \downarrow\ 1\ 0\ 1$ is equal to 5. Formally, $R\downarrow X$ is equal to $+/W\times X$, where W is the weighting vector determined as follows: $W[\rho W]$ is equal to 1, and $W[I-1]$ is equal to $R[I]\times W[I]$. For example, if R is $24\ 60\ 60$, then W is $3600\ 60\ 1$.

The result is a scalar.

The arguments R and X must be of the same dimension, except that either may be a scalar (or one-element vector). For example, $10\ \downarrow\ 1\ 7\ 7\ 6$ is equal to 1776. The arguments are not restricted to integer values. If X is a scalar, then $X\downarrow C$ is the value of a polynomial in X with coefficients C , arranged in order of descending powers of X .

The decode function is commonly applied in work with fixed-base number systems and is often called the base value function.

Encode. The encode function $R\uparrow N$ denotes the representation of the scalar N in the base- R number system. Thus, if $Z\leftarrow R\uparrow N$, then $(\times/R)\downarrow N-R\downarrow Z$ is equal to zero. For example, $2\ 2\ 2\ 2\ \uparrow\ 5$ is 0 1 0 1 and $2\ 2\ 2\ 2\ \uparrow\ 5$ is 1 0 1 and $2\ 2\ \uparrow\ 5$ is 0 1. The dimension of $R\uparrow N$ is the dimension of R . The encode function is also called representation.

Index of. If V is a vector and S is a scalar, then $J\leftarrow V\downarrow S$ yields the position of the earliest occurrence of S in V . If S does not equal any element of V , then J has the value $(\downarrow 1)+\rho V$. Clearly, this value depends, as does any result of this function, on the index origin, and is one greater than the largest permissible index of V .

If S is a vector, then J is a vector such that $J[I]$ is the index in V of $S[I]$. For example:

```
'ABCDEFGH'\downarrow 'GAFFE'
7 1 6 6 5
```

If X is a numerical vector, then the expression $X\downarrow\uparrow/X$ yields the index of the (first) maximum element in X . For example, if X is the vector 8 3 5 13 2 7 9, then \uparrow/X is 13 and $X\downarrow\uparrow/X$ is 4.

The result in every case has the same dimensions as the righthand argument of \downarrow . For example, if $Z\leftarrow V\downarrow S$, and S is a matrix, then $Z[I;J]$ is equal to $V\downarrow S[I;J]$.

Membership. The function $X\epsilon Y$ yields a logical array of the same dimension as X . Any particular element of $X\epsilon Y$ has the value 1 if the corresponding element of X belongs to Y , that is, if it occurs as some element of Y . For example, $(\downarrow 7)\epsilon 3\ 5$ is equal to 0 0 1 0 1 0 0 and $'ABCDEFGH'\epsilon 'COFFEE'$ equals 0 0 1 0 1 1 0 0.

If the vector U represents the universal set in some finite universe of discourse, then $U\epsilon A$ is the characteristic of the set A , and the membership function is therefore also called the characteristic function.

The size of the result of the function ϵ is determined by the size of the left argument, whereas the size of the result of the dyadic function \downarrow is determined by the size of the right argument. However, the left arguments of both frequently play the role of specifying the universe of discourse.

Take and drop. If V is a vector and S is a scalar between 0 and ρV , then $S \uparrow V$ takes the first S components of V . For example, if $V \leftarrow 17$, then $3 \uparrow V$ is 1 2 3 and $0 \uparrow V$ is 10, and $8 \uparrow V$ yields a domain error.

If S is chosen from the set $-1 \rho V$, then $S \uparrow V$ takes the last $|S|$ elements of V . For example, $-3 \uparrow V$ is 5 6 7.

If A is an array, then $W \uparrow A$ is valid only if W has one element for each dimension of A , and $W[I]$ determines what is to be taken along the I th coordinate of A . For example, if $A \leftarrow 3 \ 4 \rho 12$, then $2 \ -3 \uparrow A$ is the matrix

```
2 3 4
6 7 8
```

The function drop (\downarrow) is defined analogously, except that the indicated number of elements are dropped rather than taken. For example, $-1 \ 1 \downarrow A$ is the same matrix as the one displayed in the preceding paragraph.

The rank of the result of the take and drop functions is the same as the rank of the right argument. The take and drop functions are similar to the transpose in that the left argument concerns the dimension vector of the right argument.

Grade up and down. The function $\uparrow V$ produces the permutation which would order V , that is $V[\uparrow V]$ is in ascending order. For example, if V is the vector 7 1 16 5 3 9, then $\uparrow V$ is the vector 2 5 4 1 6 3, since 2 is the index of the first in rank, 5 is the index of the second in rank, and so on. The symbol \uparrow is formed by overstriking $|$ and Δ .

If P is a permutation vector, then $\uparrow P$ is the permutation inverse to P . If a vector D contains duplicate elements, then the ranking among any set of equal elements is determined by their positions in D . For example, $\uparrow 5 \ 3 \ 7 \ 3 \ 9 \ 2$ is the vector 6 2 4 1 3 5.

The right argument of \uparrow may be any array A of rank greater than zero, and the coordinate J along which the grading is to be applied may be indicated by the usual notation $\uparrow[J]A$. The form $\uparrow A$ applies as usual to the last coordinate. The result of $\uparrow A$ is of the same dimension as A .

The grade down function \downarrow is the same as the function \uparrow except that the grading is determined in descending order. Because of the treatment of duplicate items, the expression $\wedge/(\downarrow V) = \phi \downarrow V$ has the value 1 if and only if the elements of the vector V are all distinct.

Deal. The function $M?N$ produces a vector of dimension M obtained by making M random selections, without replacement, from the population $!N$. In particular, $N?N$ yields a random permutation of order N . Both arguments are limited to scalars or one-element arrays.

Comments. The lamp symbol \circ , formed by overstriking \circ and \circ , signifies that what follows it is a comment, for illumination only and not to be executed; it may occur only as the first character in a statement, but may be used in defined functions.

MULTIPLE SPECIFICATION

Specification (\leftarrow) may (like any other function) occur repeatedly in a single statement. For example, the execution of the statement $Z\leftarrow X\times A\leftarrow 3$ will assign to A the value 3, then multiply this assigned value of A by X and assign the resulting value to Z .

Multiple specification is useful for initializing variables. For example:

```
X←Y←1+Z←0
```

sets X and Y to 1 and Z to 0.

A branch may occur in a statement together with one or more specifications, provided that the branch is the last operation to be executed (i.e., the leftmost). For example, the statement $\rightarrow S\times !N>I\leftarrow I+1$ first augments I , and then branches to statement S if N exceeds the new value of I .

In the expression $Z\leftarrow(A+B)\times(C+D)$ it is immaterial whether the left or the right argument of the \times is evaluated first, and hence no order is specified. The principle of no specified order in such cases is also applied when the expressions include specification. Since the order here is sometimes material, there is no guarantee which of two or more possible results will be produced.

Suppose, for example, that A is assigned the value 5 and the expression $Z\leftarrow(A+3)\times A$ is then executed. If the left argument of \times is executed first, then A is assigned the value 3, the right argument then has the new value 3 and Z is finally assigned the value 9. If, on the other hand, the right argument is evaluated first it has the value 5 initially assigned to A , the value 3 is then assigned to A and multiplied by the 5 to yield a value of 15 to be assigned to Z .

SYSTEM DEPENDENT FUNCTIONS

There are three main types of information about the state of the system which are of value to the user:

1. general information common to all users, such as date, time of day, and the current number of terminals connected to the system.
2. information specific to the particular work session, such as the time of sign-on, the central computer time used, and the total keying time.
3. information specific to the active workspace, such as the amount of storage available, and the condition of the state indicator.

This information is provided by a single family of functions denoted by I (formed by overstriking τ , and \perp), and called the I-Beam functions. The individual member function is selected by the argument as shown in Table 3.10. Times are all in units of one-sixtieth of a second, the date is given as a six-digit integer in which the successive digit pairs specify the month, day, and year, and the available storage is given in bytes.

The byte is a unit of storage equal to 8 binary digits. A variable requires for storage a small number of bytes of overhead, plus a certain number of bytes per element depending upon the form of its representation: 1 if the elements are characters, 0.125 if the elements are logical, 4 if the elements are integers less than $2 \cdot 31$ in magnitude, and 8 for other numbers.

In designing an algorithm for a particular purpose, it frequently happens that one may trade time for space; that is, an algorithm which requires little computer time may require more storage space for intermediate results, and an algorithm which requires little storage may be less efficient in terms of time. Hence, the information provided by the functions $r21$ (computer time used) and $r22$ (available storage space) may be helpful in designing algorithms. For example, the function *TIME* of Appendix B can be used to determine the computer time used in the execution of a function.

Moreover, since the functions $r21$ and $r22$ can, like all of the I-beam functions, be used within a defined function, they can be used to make the execution dependent upon the space available or the computer time used.

X	Definition of iX
19	Accumulated keying time (time during which the keyboard has been unlocked awaiting entries) during this session.
20	The time of day.
21	The central computer time used in this session.
22	The amount of available space (in bytes).
23	The number of terminals currently connected.
24	The time at the beginning of this session.
25	The date.
26	The first element of the vector $i27$.
27	The vector of statement numbers in the state indicator.

NOTES

1. All times in 1÷60 seconds
2. Date is represented by a 6-digit integer; successive digit pairs represent month, day, and year.
3. $i27$ yields a vector; all other results are scalars.

Table 3.10: SYSTEM DEPENDENT FUNCTIONS

Keying time is defined as the total accumulated time since sign-on during which the keyboard has been unlocked awaiting entry. The associated function ($i19$) may be used in conjunction with \square or \square input to determine the amount of time taken by a student in responding to a question. The following example shows the definition and use of a multiplication drill which tells the student how long he has taken (in whole minutes and seconds) to answer each question:

```

VMULTDRILL N;X;Y;TIME
[1]   $\square \leftarrow Y \leftarrow ?N$ 
[2]   $TIME \leftarrow i19$ 
[3]   $X \leftarrow \square$ 
[4]   $TIME \leftarrow (i19) - TIME$ 
[5]   $\rightarrow 8 \times i X \neq \times / Y$ 
[6]  'TIME: ';  $2 \uparrow 60$  60  $60 \uparrow TIME$ 
[7]   $\rightarrow 1$ 
[8]  'WRONG, TRY AGAIN'
[9]   $\rightarrow 3 \nabla$ 
MULTDRILL 12 12
6 3
 $\square$ :
18
TIME: 0 3
4 5
30
WRONG, TRY AGAIN
 $\square$ :
 $\rightarrow$ 

```

Such a drill could be expanded to accumulate statistics of the student's response times or to use some function of the response times to control the difficulty of the questions posed.

Since times are expressed in units of 1/60 seconds, the time in hours, minutes, and seconds can be determined by an expression of the form $3 \uparrow 24 \ 60 \ 60 \ 60 \uparrow I20$. Similarly, a 3-element vector representing the date can be obtained from the expression $(3 \rho 100) \uparrow I25$.

The expression $I27$ provides the vector of statement numbers in the state indicator, with the first position occupied by the number of the statement on which the innermost function is suspended. If no functions are suspended, the vector $I27$ is empty.

The expression $I26$ yields a scalar which is the first element of $I27$. It is therefore equal to the number of the statement being, or about to be, executed and is particularly useful in branches. For example, $\rightarrow N + I26$ causes a forward jump of N statements. Moreover, entering $\rightarrow I26$ is a safe way to resume execution without having to read and enter the statement number printed at the point of the last suspension. It is even more convenient to resume by entering $\rightarrow C$, after first defining the function C as follows:

```
      VZ←C
[1]   Z←(I27)[2]V
```


PART 4

LIBRARY FUNCTIONS

A user may load or copy functions from any workspace for which he knows the library number and workspace name (and password, if any). Moreover a listing of the workspaces in Library *N* can be obtained by the command `)LIB N` for any public library, i.e., for any library whose number is below 1000.

A public library may be used for the casual sharing of functions among a group of co-workers. When intended for more general use, a library function should be thoroughly tested and well-documented, and should incorporate messages for the guidance of the user. It is therefore good practice to restrict certain of the public libraries to such functions as are of general interest and have passed appropriate acceptance tests.

In the APL\360 system as distributed, Library 1 is restricted in this manner. This section treats each of the workspaces in this library by loading each and displaying the descriptions contained in the workspaces themselves. Further information on the functions in each workspace can (except in the case of the locked functions in *WSFNS*) be obtained by displaying the function definitions.

```
)LOAD 1 ADVANCEDEX
ADVANCEDEX SAVED 07/14/68 16.53.19
)FNS
AH      ASSOC  BIN    COMB    DTH     ENTER   F       FC
GC      GCD   GCV    HILB   HTD     IN      INV    INVVP
IN1     LFC   LOOKUP PALL   PER     PERM   PO     POL
POLY    POLYB RESET  TIME   TRUTH  ZERO
```

DESCRIBE

EACH OF THE VARIABLES OF THIS WORKSPACE WHICH BEGINS WITH THE LETTER D IS THE DESCRIPTION OF THE FUNCTION WHOSE NAME IS OBTAINED BY REMOVING THE D. FOR FURTHER DETAILS SEE APPENDIX B OF THE APL\360 MANUAL.

```

)LOAD 1 PLOTFORMAT
PLOTFORMAT SAVED 07/20/68 31.07.27
)FNS
AND DESCRIBE DFT EFT PLOT VS

DESCRIBE

```

THE FUNCTIONS INCLUDED IN THIS WORKSPACE ARE LISTED BELOW:

<u>SYNTAX</u>	<u>DESCRIPTION</u>
Z←A AND B	ESSENTIALLY A COLUMN-CATENATOR, WITH SOME EXTRA EFFECTS WHEN THE ARGUMENTS ARE NOT MATRICES. THIS FUNCTION IS DESIGNED TO BE USED EITHER INDEPENDENTLY, OR IN CONJUNCTION WITH VS. IT PROVIDES A CONVENIENT WAY OF FORMING INPUT TO DFT AND EFT.
Z←A DFT B	FORMS FIXED-POINT OUTPUT. MORE DETAILED DIRECTIONS CAN BE FOUND IN THE VARIABLE HOWFORMAT.
Z←A EFT B	FORMS EXPONENTIAL OUTPUT. MORE DETAILED DIRECTIONS CAN BE FOUND IN THE VARIABLE HOWFORMAT.
A PLOT B	GRAPHS ONE OR MORE FUNCTIONS SIMULTANEOUSLY. DIRECTIONS FOR USING PLOT CAN BE FOUND IN THE VARIABLE HOWPLOT.
Z←A VS B	ESSENTIALLY A COLUMN-CATENATOR, SIMILAR TO AND, EXCEPT THAT THE RIGHT-HAND ARGUMENT MUST BE OF RANK ≤ 1. IT IS DESIGNED PRIMARILY TO PROVIDE CONVENIENT FORMATION OF INPUT TO PLOT FUNCTION. WHETHER USED BY ITSELF OR WITH AND, VS WILL CAUSE ITS RIGHT ARGUMENT TO APPEAR AS THE LEFTMOST COLUMN OF THE RESULTANT ARRAY. (THE RESULTANT WILL BE AN ARRAY OF RANK THREE, CONSISTING OF A SINGLE PLANE).

BOTH AND AND VS WORK WITH EITHER 1 OR 0-ORIGIN INDEXING.

HOWFORMAT

THE FUNCTIONS DFT AND EFT WILL ARRAY NUMBERS IN DECIMAL AND EXPONENTIAL FORM, RESPECTIVELY, FOR TABULAR OUTPUT. THEY MAY BE USED TO GENERATE IMMEDIATE OUTPUT, OR TO STORE AN IMAGE FOR LATER PRINTING. THE TWO FORMS ARE:

PATTERN DFT TABLE
PATTERN EFT TABLE

AND

IMAGE←PATTERN DFT TABLE
IMAGE←PATTERN EFT TABLE

THESE FUNCTIONS WORK PROPERLY ONLY WITH 1-ORIGIN INDEXING.

RIGHT ARGUMENT: AN ARRAY TO BE FORMED.

IT MUST BE NUMERICAL, AND OF RANK ≤ 3 . THE FIRST PLANE OF A 3-DIMENSIONAL ARRAY WILL BE TREATED AS A MATRIX, AND ALL OTHER PLANES WILL BE DISREGARDED. ARRAYS OF HIGHER RANK WILL BE SIGNALLED AS A 'RANK PROBLEM.'

LEFT ARGUMENT: ONE OR MORE INTEGERS TO CONTROL THE FORMAT. FRACTIONAL NUMBERS WILL BE SIGNALLED AS A 'DOMAIN PROBLEM.'

A SINGLE INTEGER:

DFT: SPECIFIES THE NUMBER OF DIGITS TO THE RIGHT OF THE DECIMAL POINT IN DECIMAL FORMAT.

EFT: SPECIFIES THE NUMBER OF SIGNIFICANT DIGITS IN EXPONENTIAL FORMAT. ONE DIGIT ALWAYS APPEARS TO THE LEFT OF THE DECIMAL POINT. COLUMNS WILL BE SPACED UNIFORMLY, WITH SPACING SUCH THAT THERE WILL BE TWO SPACES BETWEEN THE CLOSEST NUMBERS.

A PAIR OF INTEGERS: THE FIRST SPECIFIES THE TOTAL NUMBER OF SPACES TO BE ALLOCATED TO EACH COLUMN, AND THE SECOND IS USED AS ABOVE.

DFT: THE FIRST NUMBER MUST BE AT LEAST TWO LARGER THAN THE SECOND.

EFT: THE FIRST NUMBER MUST BE AT LEAST SIX LARGER THAN THE SECOND. IF THE LEFT NUMBER IS TOO SMALL, THIS WILL BE SIGNALLED AS A 'DOMAIN PROBLEM.'

MORE THAN ONE PAIR OF INTEGERS: THERE MUST BE ONE PAIR FOR EACH COLUMN OF OUTPUT (OR EACH ELEMENT OF A VECTOR). EACH PAIR WILL BE INTERPRETED AS ABOVE, AND WILL APPLY TO THE LAYOUT OF THE CORRESPONDING COLUMN. IF THE NUMBER OF PAIRS DOES NOT MATCH THE NUMBER OF COLUMNS, THIS WILL BE SIGNALLED AS A 'LENGTH PROBLEM.'

HOWPLOT

THE FUNCTION PLOT WILL GRAPH ONE OR MORE FUNCTIONS SIMULTANEOUSLY, AUTOMATICALLY SCALING THE VALUES TO FIT APPROXIMATELY WITHIN SCALE DIMENSIONS SPECIFIED BY THE USER. IT WILL WORK ONLY IN 1-ORIGIN INDEXING.

THE FORM IN WHICH PLOT IS USED IS:

SCALESIZE PLOT FUNCTION

LEFT ARGUMENT: ONE OR TWO NUMBERS.

THE FIRST NUMBER SPECIFIES THE APPROXIMATE SIZE OF THE VERTICAL AXIS AND THE SECOND NUMBER DOES THE SAME FOR THE HORIZONTAL AXIS.

IF ONLY ONE NUMBER IS SUPPLIED, IT IS APPLIED TO BOTH AXES.

THERE IS NO BUILT-IN LIMIT TO THE DIMENSIONS, AND A HORIZONTAL AXIS LARGER THAN THE WORKSPACE WIDTH WILL CAUSE SOME POINTS TO BE PRINTED ON THE NEXT LOWER LINE.

RIGHT ARGUMENT: A RECTANGULAR ARRAY WITH RANK ≤ 3 .

SCALAR: WILL BE TREATED AS A VECTOR OF LENGTH ONE.

VECTOR: WILL BE PLOTTED AS ORDINATE AGAINST ITS OWN INDICES AS ABSCISSA.

MATRIX: THE LEFTMOST COLUMN WILL BE TAKEN AS THE ABSCISSA AND ALL OTHER COLUMNS WILL BE PLOTTED AS ORDINATES. A DIFFERENT PLOTTING SYMBOL UP TO THE NUMBER OF SYMBOLS AVAILABLE WILL BE USED FOR EACH COLUMN. IN CASE TWO ORDINATES HAVE A COMMON POINT, THE SYMBOL FOR THE COLUMN FURTHEST TO THE RIGHT WILL BE USED.

3-DIMENSIONAL ARRAY: THE FIRST PLANE WILL BE PLOTTED AS A MATRIX, AND ALL OTHER PLANES WILL BE DISREGARDED.

AUXILIARY FUNCTIONS: THE FUNCTIONS AND AND VS CAN BE USED TO GENERATE THE RIGHT ARGUMENT IN THE PROPER FORM FOR PLOT. FOR EXAMPLE:

20 PLOT Z AND Y VS X

PLOT CHARACTERS: THE SYMBOLS USED ARE ASSIGNED TO THE VARIABLE PC IN LINE 1 OF PLOT. THE ALPHABET SUPPLIED IS 'O*O∇Δ□'. THIS ALPHABET MAY BE EXTENDED AND MODIFIED AS DESIRED, USING THE NORMAL FUNCTION-EDITING PROCEDURES: EITHER CHANGE LINE 1 OF THE FUNCTION, OR DELETE IT AND INDEPENDENTLY SPECIFY A VALUE FOR PC.

HISTOGRAMS: PLOT CAN BE USED TO GENERATE HISTOGRAMS BY SETTING THE VARIABLE HS TO 1 IN LINE 2 OF THE FUNCTION. ALTERNATIVELY, LINE 2 CAN BE DELETED, AND HS CAN BE SET EXTERNALLY.

```
)LOAD 1 APLCOURSE
APLCOURSE SAVED 07/19/68 25.58.06
)FNS
B1X CHECK DESCRIBE DIM DRILL DYAD1 DYAD2
EASY EASYDRILL FORM FUNDRILL GET INPUT
INTER LOG QUES RANDOM REDSCAPATCH REPP
SETPARAMETERS TEACH TRACE
```

DESCRIBE

THE MAIN FUNCTIONS IN THIS LIBRARY WORKSPACE ARE:

TEACH
EASYDRILL

ALL OTHER FUNCTIONS ARE SUBFUNCTIONS AND ARE NOT SELF-CONTAINED.

SYNTAX

DESCRIPTION

TEACH AN EXERCISE IN APL FUNCTIONS USING SCALARS AND VECTORS. THE FUNCTION PRINTS OUT THE CHOICES AND OPTIONS AVAILABLE. EXAMPLES ARE SELECTED AT RANDOM WITH A RANDOM STARTING POINT.

EASYDRILL THIS IS THE SAME AS TEACH EXCEPT THAT THE PROBLEMS SELECTED ARE GENERALLY SIMPLER IN STRUCTURE. PROBLEMS INVOLVING VECTORS OF LENGTH ZERO OR ONE ARE EXCLUDED.

NOTE: FOR EITHER FUNCTION, A RESPONSE OF - PLEASE - WILL DISCLOSE THE PROPER ANSWER. A RESPONSE OF - STOP - WILL TERMINATE THE DRILL.

TEACH

ARE YOU ALREADY FAMILIAR WITH THE INSTRUCTIONS? (TYPE Y FOR YES AND N FOR NO.)

N

THIS IS AN EXERCISE IN SIMPLE APL EXPRESSIONS. YOU WILL FIRST HAVE THE OPPORTUNITY TO SELECT THE FEATURES YOU WISH TO BE DRILLED IN. THE EXERCISE THEN BEGINS. FOR EACH PROBLEM YOU MUST ENTER THE PROPER RESULT. ANSWERS WILL CONSIST OF SCALAR INTEGERS IF EXERCISES WITH VECTORS ARE NOT DESIRED; OTHERWISE ANSWERS WILL CONSIST OF SCALARS OR VECTORS. A VECTOR OF LENGTH ZERO REQUIRES THE RESPONSE 10, A VECTOR OF LENGTH ONE REQUIRES THE RESPONSE X, WHERE X IS THE VALUE OF THE ELEMENT. YOU HAVE THREE TRIES FOR EACH PROBLEM. TYPE STOP AT ANY TIME TO TERMINATE THE EXERCISE AND PRODUCE A RECORDING OF YOUR PERFORMANCE. TYPING STOPSHORT WILL TERMINATE THE EXERCISE BUT WILL NOT PRODUCE A RECORD OF PERFORMANCE. TYPING PLEASE FOR ANY PROBLEM WILL LET YOU PEEK AT THE ANSWERS. TYPE Y UNDER EACH FUNCTION FOR WHICH YOU WANT EXERCISE:

SCALAR DYADIC FUNCTIONS

+ - × ÷ * [| < = > ≠ ! | ^ ∨ ⊕ ⊗ * * *

YYYYY Y

SCALAR MONADIC FUNCTIONS

+ - × ÷ [[! | ~

YY

TYPE Y IF EXERCISES ARE TO USE VECTORS, N OTHERWISE

N

$^{-6} \times ^{-3}$

□:

18

$[-2.5$

□:

$^{-2}$

TRY AGAIN

□:

$^{-3}$

0+7

□:

STOPSHORT

```
)LOAD 1 WSFNS
WSFNS SAVED 07/20/68 31.25.23
```

```
)FNS
DELAY DESCRIBE DIGITS ORIGIN SETLINK SFEI WIDTH
DESCRIBE
```

THE FUNCTIONS ORIGIN, WIDTH, AND DIGITS ARE EACH SIMILAR TO THE COMMAND OF THE SAME NAME, EXCEPT THAT EACH IS A FUNCTION RATHER THAN A COMMAND AND MAY THEREFORE BE USED WITHIN OTHER FUNCTIONS. EACH HAS AN EXPLICIT RESULT WHICH IS THE PREVIOUS VALUE OF THE RELEVANT SYSTEM PARAMETER.

FOR EXAMPLE, THE FOLLOWING FUNCTION:

```
VF X
[1] X←ORIGIN X
[2] G
[3] X←ORIGIN X V
```

WILL EXECUTE THE FUNCTION G WITH WHATEVER INDEX ORIGIN IS SPECIFIED BY THE ARGUMENT OF F, AND WILL RESTORE THE INDEX ORIGIN TO THE VALUE THAT IT HAD BEFORE THE EXECUTION OF F.

THE FOLLOWING FUNCTIONS ARE ALSO AVAILABLE:

<u>SYNTAX</u>	<u>DESCRIPTION</u>
Z←SETLINK X	SETS THE VALUE OF THE LINK IN THE CHAIN OF NUMBERS GENERATED IN THE USE OF THE ROLL AND DEAL FUNCTIONS. THE EXPLICIT RESULT PRODUCED BY SETLINK IS THE PREVIOUS VALUE OF THE LINK. THE RESULTS PRODUCED BY THE ROLL AND DEAL FUNCTIONS ARE NOT THE LINKS THEMSELVES, BUT RATHER SOME FUNCTION OF THEM. THE LENGTH OF THE CHAIN (BEFORE REPETITION) IS $\bar{1}+2*31$.
DELAY X	DELAYS EXECUTION FOR X SECONDS.
SFEI X	SETS THE SIGNAL MESSAGE FOR EVALUATED INPUT (SEE THE SECTION ON INPUT AND OUTPUT, PART 3 OF THE APL\360 USER'S MANUAL). THE ARGUMENT MUST BE A LINE OF NO MORE THAN 7 CHARACTERS.

)LOAD 1 TYPEDRILL
TYPEDRILL SAVED 07/14/68 19.42.16

)FNS
DESCRIBE IN INSTRUCTIONS MATCH PRT' QUERY
STATISTICS TIME TYPEDRILL WS

DESCRIBE

THE MAIN FUNCTION IN THIS WORKSPACE IS TYPEDRILL; ALL OTHERS ARE SUBFUNCTIONS. TO USE IT, SIMPLY ENTER

TYPEDRILL

TYPEDRILL IS A TIMED TYPING EXERCISE. THE SYSTEM RESPONDS WITH THE STATEMENT 'YOU ARE IN CONTROL STATE'. FOUR COMMANDS ARE AT YOUR DISPOSAL: ENTER, DRILL, STAT, AND STOP. ENTERING ONE OF THEM BRINGS YOU INTO THAT STATE:

ENTER: YOU MAY ENTER ONE-LINE SENTENCES OR EXPRESSIONS ON WHICH YOU WISH TO BE DRILLED. ENTERING A BLANK LINE (CARRIAGE RETURN ONLY) RETURNS YOU TO THE CONTROL STATE.

DRILL: ONE OF THE LINES ENTERED VIA THE ENTER STATE IS SELECTED AT RANDOM AND PRINTED. YOU ARE THEN EXPECTED TO ENTER THE SAME LINE. IF IT IS CORRECT, THE TIME TAKEN IS PRINTED (IN SECONDS), IF NOT YOU ARE ASKED TO RETYPE IT. A BLANK LINE CAUSES RETURN TO THE CONTROL STATE.

STAT: THE ACCUMULATED STATISTICS ARE PRINTED. THE HORIZONTAL AXIS SHOWS THE TRIAL NUMBERS AND THE VERTICAL SHOWS THE TIME IN SECONDS. A VERTICAL ARROW INDICATES THAT THE TIME EXCEEDED THE LIMITS OF THE GRAPH. THE RETURN TO THE CONTROL STATE IS AUTOMATIC.

STOP: STOPS THE DRILL AND PRINTS THE STATISTICS.

TYPEDRILL
CONTROL WORDS ARE: ENTER, DRILL, STAT, AND STOP.

YOU ARE IN CONTROL STATE

ENTER

*NOW IS THE TIME FOR ALL GOOD MEN TO COME TO
I SING OF OLAF GLAD AND BIG*

X←|P×Q+Y×R≤5

YOU ARE IN CONTROL STATE

DRILL

*NOW IS THE TIME FOR ALL GOOD MEN TO COME TO
NOW IS THE TIME OFR ALL GOOD MEN TO COME TO*

^

*NOW IS THE TIME FOR ALL GOOD MEN TO COME TO
16.9*

X←|P×Q+Y×R≤5

X←|P×Q+Y×R≤5

19.9

I SING OF OLAF GLAD AND BIG

YOU ARE IN CONTROL STATE

STOP

Appendix A

SAMPLE TERMINAL SESSION

)1776
010) 19.32.36 07/03/68 JANET

A P L \ 3 6 0

FUNDAMENTALS

12	3×4	Entry automatically indented
	X←3×4	Response not indented
	X	X is assigned value of the expression
12	Y← ⁻ 5	Value of X typed out
	X+Y	Negative sign for negative constants
7	144E ⁻ 2	Exponential form of constant
1.44	P←1 2 3 4	Four-element vector
	P×P	Functions apply element by element
1 4	9 16	
	P×Y	Scalar applies to all elements
-5	⁻ 10 ⁻ 15 ⁻ 20	
	Q←'CATS'	Character constant (4-element vector)
	Q	
CATS	YZ←5	Multi-character names
	YZ1←5	
	YZ+YZ1	
10	3+4×5+6	Correction by backspace and linefeed
	v	
	+5+6	
18	X←3	
	Y←4	
	(X×Y)+4	
16	X×Y+4	Executed from right to left
24		

```

      X Y
SYNTAX ERROR
      X Y
      ^
      XY
VALUE ERROR
      X
      ^

      4×3[5.1
20.4
      (4×3)[5.1
12
      4×[5.1
24
      X←15
      X
1 2 3 4 5
      10

      Y←5-X
      Y
4 3 2 1 0
      X[Y
4 3 3 4 5
      X≤Y
1 1 0 0 0
      01
3.141592654
      0÷1 2
3.141592654 1.570796327
      X←45 90
      0X÷180
0.7853981634 1.570796327
      101
0.8418709848
      201 2
0.5403023059 -0.4161468365
      301
1.557407725
      -301
0.7853981634
      30-3017
1 2 3 4 5 6 7
      Y←1 2
      40Y
1.414213562 2.236067977
      00÷Y
0 0.8660254038
      701 2
0.761594156 0.9640275801
      -70701 2
1 2

```

Entry of invalid expression
Shows type of error committed
Retypes invalid statement with
caret where execution stopped
Multi-character name (not X×Y)

XY had not been assigned a value

SCALAR FUNCTIONS

Dyadic maximum

Monadic ceiling

Index generator function

Empty vector
prints as a blank line
All scalar functions extend
to vectors

Relations produce
logical (0 1) results
Pi×1

Pi÷1 2

Conversion of X to radians

Sin 1

Cos 1 2

Tan 1

Arctan 1

Tan Arctan 1 2 3 4 5 6 7

(1+Y*2)*.5

(1-÷Y*2)*.5

Tanh 1 2

Arctanh Tanh 1 2

DEFINED FUNCTIONS

<pre> [1] ∇Z←X F Y [2] Z←((X*2)+Y*2)*.5 [2] ∇ 3 F 4 5 P←7 Q←(P+1)F P-1 Q 10 4×3 F 4 20 ∇B←G A [1] B←(A>0)-A<0 [2] ∇ G 4 1 G ^6 ^-1 X←^6 G X ^-1 ∇H A [1] P←(A>0)-A<0 [2] ∇ H ^6 P ^-1 Y←H ^6 VALUE ERROR Y←H ^6 ^ ∇Z←FAC N;I [1] Z←1 [2] I←0 [3] L1:I←I+1 [4] →0×1I>N [5] Z←Z×I [6] →L1 [7] ∇ FAC 3 6 FAC 5 120 TΔFAC←3 5 X←FAC 3 FAC[3] 1 FAC[5] 1 FAC[3] 2 FAC[5] 2 FAC[3] 3 FAC[5] 6 FAC[3] 4 TΔFAC←0 </pre>	<pre> Header (2 args and result) Function body Close of definition Execution of dyadic function F Use of F with expressions as arguments G is the signum function A and B are local variables Like G but has no explicit result P is a global variable H has no explicit result and hence produces a value error when used to right of assignment FAC is the factorial function L1 becomes 3 at close of def Branch to 0 (out) or to next Branch to L1 (that is, 3) Set trace on lines 3 and 5 of FAC Trace of FAC Reset trace control </pre>
--	---

MECHANICS OF
FUNCTION DEFINITION

<pre> ∇G←M GCD N [1] G←N [2] M←M N [3] →4×M≠0 [4] [1]G←M [2] [4]N←G [5] [1□] [1] G←M [1] [□] ∇ G←M GCD N [1] G←M [2] M←M N [3] →4×M≠0 [4] N←G ∇ [5] →1 [6] ∇ 36 GCD 44 4 ∇GCD [6] [4.1]M,N [4.2] [□] ∇ G←M GCD N [1] G←M [2] M←M N [3] →4×M≠0 [4] N←G [4.1] M,N [5] →1 ∇ [6] ∇ 36 GCD 44 8 36 4 8 4 ∇GCD[□]∇ ∇ G←M GCD N [1] G←M [2] M←M N [3] →4×M≠0 [4] N←G [5] M,N [6] →1 ∇ ∇GCD [7] [5] ^ ∇ </pre>	<pre> Greatest common divisor function based on the Euclidean algorithm Correction of line 1 Resume with line 4 Display line 1 Display entire GCD Function Close of display, not close of def Enter line 5 Close of definition Use of GCD 4 is GCD of 36 and 44 Reopen def (Use ∇ and name only) Insert between 4 and 5 Display entire function Fraction stays until close of def End of display Close of definition Iterations printed by line 5 (was line 4.1) Final result Reopen, display, and close GCD Line numbers have been reassigned as integers Close (Even number of ∇'s in all) Reopen definition of GCD Delete line 5 by linefeed Close definition </pre>
--	---

<pre> ∇Z←ABC X [1] Z←(33×Q)+(R×5)-6 [2] [1□9] [1] Z←(33×Q)+(R×5)-6 / 1 /1 [1] Z←(3×Q)+(T×5)-6 [2] ∇ FAC 5 120)ERASE FAC FAC 5 SYNTAX ERROR FAC 5 ^ ∇Z←BIN N [1] LA:Z←(Z,0)+0,Z [2] →LA×N≥ρZ∇ BIN 3 VALUE ERROR BIN[1] LA:Z←(Z,0)+0,Z ^ Z←1 →1 1 3 3 1 BIN 4 VALUE ERROR BIN[1] L1:Z←(Z,0)+0,Z ^ ∇BIN[.1]Z←1∇)SI BIN[1] * →1 1 4 6 4 1 ∇BIN[□]∇ ∇ Z←BIN N [1] Z←1 [2] LA:Z←(Z,0)+0,Z [3] →LA×N≥ρZ ∇ SΔBIN+2 Q←BIN 3 BIN[2] Z 1 →2 BIN[2] →2 BIN[2] →0 </pre>	<pre> A function to show line editing A line to be corrected Initiate edit of line 1 Types line, stops ball under 9 Slash deletes, digit inserts spaces Ball stops at first new space. Then enter) T FAC still defined Erase function FAC Function FAC no longer exists An (erroneous) function for binomial coefficients Suspended execution Assign value to Z Resume execution Binomial coefficients of order 3 Same error (local variable Z does not retain its value) Insert line to initialize Z Display state indicator Suspended on line 1 of BIN Resume execution (BIN now correct) Display revised function and close definition Set stop on line 2 Execute BIN Stop due to stop control Display current value of Z Resume execution Stop again on next iteration Resume Stop again Branch to 0 (terminate) </pre>
--	--

INPUT AND OUTPUT

```

VMULTDRILL N;Y;X
[1] Y←?N
[2] Y
[3] X←□
[4] →0×1X='S'
[5] →1X=×/Y
[6] 'WRONG, TRY AGAIN'
[7] →3∇
MULTDRILL 12 12

```

2 10

□:

37

WRONG, TRY AGAIN

□:

20

6 7

□:

'S'

∇Z←ENTERTEXT

[1] Z←''

[2] D←ρZ

[3] Z←Z,□

[4] →2×D≠ρZ

[5] ∇

Q←ENTERTEXT

THIS IS ALL

CHARACTER INPUT

Q

THIS IS ALL CHARACTER INPUT

N←5

'NOTE: ₁' ; N ; ' IS ' ; ₁N

NOTE: ₁5 IS 1 2 3 4 5

A multiplication drill
 ρN random integers
 Print the random factors
 Keyboard input
 Stop if entry is the letter S
 Repeat if entry is correct product
 Prints if preceding branch fails
 Branch to 3 for retry
 Drill for pairs in range 1 to 12

Indicates that keyboard entry
 is awaited

Entry of letter S stops drill
 Example of character (□) input
 Make Z an empty vector
 D is the length of Z
 Append character keyboard entry
 Branch to 2 if length increased
 (i.e., entry was not empty)

Keyboard
 entries
 Empty input to terminate
 Display Q

Mixed output statement

RECTANGULAR ARRAYS

```

P←2 3 5 7
ρP
4
T←'OH MY'
ρT
5
P,P
2 3 5 7 2 3 5 7
T,T
OH MYOH MY
T,P
DOMAIN ERROR
T,P
^

```

Dimension of P

Character vector

Catenation

Characters cannot be catenated
 with numbers

<pre> M+2 3ρ2 3 5 7 11 13 M 2 3 5 7 11 13 2 4ρT OH M YOH 6ρM 2 3 5 7 11 13 ,M 2 3 5 7 11 13 P←,M P[3] 5 P[1 3 5] 2 5 11 P[13] 2 3 5 P[ρP] 13 M[1;2] 3 M[1;] 2 3 5 M[1 1;3 2] 5 3 5 3 A←'ABCDEFGHIJKLMNO P Q' A[M] BCE GKM A[M[1 1;3 2]] EC EC M[1;]←15 3 12 M 15 3 12 7 11 13 </pre>	<pre> Reshape to produce a 2×3 matrix Display of an array of rank >1 is preceded by a blank line A 2×4 matrix of characters A matrix reshaped to a vector Elements in row-major order Indexing (third element of P) A vector index The first three elements of P Last element of P Element in row 1, column 2 of M Row 1 of M Rows 1 and 1, columns 3 2 The alphabet to Q A matrix index produces a matrix result Respecifying the first row of M </pre>
--	--

```

      Q←3 1 5 2 4 6
      P[Q]
5  2  11  3  7  13
      Q[Q]
5  3  4  1  2  6
      P[3]
5
      )ORIGIN 0
WAS 1
      P[3]
7
      P[0 1 2]
2  3  5
      15
0  1  2  3  4
      )ORIGIN 1
WAS 0
      15
1  2  3  4  5

```

A permutation vector
 Permutation of P

A new permutation

Present index origin is 1

Set index origin to 0

First three elements of P

Result of index generator
 begins at origin

FUNCTIONS ON ARRAYS

```

      V←?3ρ9
      M←?3 3ρ9
      N←?3 3ρ9
      V
2  1  7
      M

7  9  4
5  8  1
1  5  7
      N
1  4  1
4  7  6
9  8  5
      M+N

8  13  5
9  15  7
10 13  12

```

Vector of 3 random integers (1-9)
 Random 3 by 3 matrix
 Random 3 by 3 matrix

Sum (element-by-element)

	$M \lceil N$	Maximum
7	9 4	
5	8 6	
9	8 7	
	$M \leq N$	Comparison
0	0 0	
0	0 1	
1	1 0	
	+ / V	Sum-reduction of V
10		
	x / V	Product-reduction
14		
	+ / [1] M	Sum over first coordinate of M
13	22 12	(down columns)
	+ / [2] M	Sum over second coordinate of M
		(over rows)
20	14 13	
	+ / M	Sum over last coordinate
20	14 13	
	⌈ / M	Maximum over last coordinate
9	8 7	
	X ← 1.5	
	+ / (1 20X) * 2	Sin squared plus Cos squared
1		
	0 / 1 2, X	Sin Cos X
0.07067822453		
	Y ← 0 / 0 2, X	(1 - (COS X) * 2) * .5
	Y	
0.9974949866		
	Y = 10X	An identity
1		
	$M + . \times N$	Ordinary matrix (+ . × inner)
		product
	79 123 81	
	46 84 58	
	84 95 66	
	$M + . \leq N$	An inner product
1	1 1	
1	1 1	
2	3 2	
	$M + . \times V$	+ . × inner product with vector
51	25 56	right argument

```

      V
2  1  7
      Vo.×15

```

Outer product (times)

```

      2  4  6  8  10
      1  2  3  4  5
      7 14 21 28 35
      Vo.≤19

```

Outer product

```

0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1
      Vo.×M

```

An outer product of rank 3

```

14 18  8
10 16  2
 2 10 14

```

A blank line between planes

```

 7  9  4
 5  8  1
 1  5  7

```

```

49 63 28
35 56  7
 7 35 49

```

MIXED FUNCTIONS

```

      Q←?10ρ5
      Q
1  4  3  4  5  4  2  1  4  2
      +/[1]Qo.=15
2  2  1  4  1
      2 1QM

```

A random 10 element vector
(range 1 to 5)

Ith element of result is number
of occurrences of the
value I in Q

Ordinary transpose of M

```

 7  5  1
 9  8  5
 4  1  7
      QM

```

Ordinary transpose of M (monadic)

```

 7  5  1
 9  8  5
 4  1  7

```

```

T←2 3 4p124
T
1 2 3 4
5 6 7 8
9 10 11 12

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

3 1 2QT
1 13
2 14
3 15
4 16

5 17
6 18
7 19
8 20

9 21
10 22
11 23
12 24

1 1QM
7 8 7
1 1 2QT

1 2 3 4
17 18 19 20
X←o(0,15)÷6
)DIGITS 4
WAS 10
Q1 2 3°.OX

0.000E0 1.000E0 0.000E0
5.000E-1 8.660E-1 5.774E-1
8.660E-1 5.000E-1 1.732E0
1.000E0 1.744E-16 5.734E15
8.660E-1 -5.000E-1 -1.732E0
5.000E-1 -8.660E-1 -5.774E-1

```

An array of rank 3

Transpose of T (dimension
of result is 3 4 2)

Diagonal of M

Diagonal section in first
two coordinates of T

Set number of output digits to 4

Table of sines, cosines, and
tangents in intervals
of 30 degrees

Q
 1 4 3 4 5 4 2 1 4 2
 $3\phi Q$
 4 5 4 2 1 4 2 1 4 3
 $-3\phi Q$
 1 4 2 1 4 3 4 5 4 2
 $0 1 2\phi[1]M$

Rotate to left by 3 places

Rotate to right by 3 places

Rotate columns by
different amounts

7 8 7
 5 5 4
 1 9 1
 $-2\phi[2]M$

Rotation of rows all
by 2 to right

9 4 7
 8 1 5
 5 7 1
 1 2 $3\phi M$

Rotation of rows

9 4 7
 1 5 8
 1 5 7
 ϕQ
 2 4 1 2 4 5 4 3 4 1

Reversal of Q

$\phi[1]M$

Reversal of M along
first coordinate

1 5 7
 5 8 1
 7 9 4
 ϕM

Reversal along last coordinate

4 9 7
 1 8 5
 7 5 1

		$U \leftarrow Q > 4$		
		U		
0	0	0 0 1 0 0 0 0 0		
		U/Q		Compression of Q by logical vector U
5				
		$(\sim U)/Q$		Compression by not U
1	4	3 4 4 2 1 4 2		
		$+U/Q$		
5				
		$1 0 1/[1]M$		Compression along first coordinate of M
7	9	4		
1	5	7		
		$1 0 1/M$		Compression along last coordinate
7	4			
5	1			
1	7			
		$(,M > 5)/,M$,M is 7 9 4 5 8 1 1 5 7
7	9	8 7		All elements of M which exceed 5
		$V \leftarrow 1 0 1 0 1$		
		$V \setminus 13$		Expansion of iota 3
1	0	2 0 3		
		$V \setminus M$		Expansion of rows of M
7	0	9 0 4		
5	0	8 0 1		
1	0	5 0 7		
		$V \setminus 'ABC'$		Expansion of literal vector inserts spaces
A	B	C		
		$10 \perp 1 7 7 6$		Base 10 value of vector 1 7 7 6
1776				
		$8 \perp 1 7 7 6$		Base 8 value of 1 7 7 6
1022				
		$(4 \rho 10) \top 1776$		4 digit base 10 representation of number 1776
1	7	7 6		
		$(3 \rho 10) \top 1776$		3 digit base 10 representation of 1776
7	7	6		
		$10 10 \top 1776$		
7	6			
		$10 \top 1776$		
6				
		$24 60 60 \perp 1 3 25$		Mixed base value of 1 3 25 (time radix)
3805				
		$24 60 60 \top 3805$		Representation of number 3805 in time radix
1	3	25		
		$2 \perp 1 0 1 1 0$		Base 2 value
22				

```

M
7 9 4
5 8 1
1 5 7
)ORIGIN 0
WAS 1
M[2;0]
1
(,M)[(ρM)12,0]
1
)ORIGIN 1
WAS 0
P
2 3 5 7 11 13
P17
4
P16
7
P14 5 6 7
7 3 7 4
Q←5 1 3 2 4
R←Q11ρQ
R
2 4 3 5 1
Q[R]
1 2 3 4 5
A←'ABCDEFGHIJKLMNO PQ'
A←A,'RSTUVWXYZ'
A
ABCDEFGHIJKLMNO PQRSTU VWXYZ
A1'C'
3
J←A1'CAT'
J
3 1 20
A[J]
CAT

```

Indexing of matrix in 0-origin.
 Note relation to indexing of
 ravel of M

Restore 1-origin

Index of 7 in vector P
 7 is 4th element of P
 6 does not occur in P, hence
 result is 1+ρP

A permutation vector
 R is the permutation inverse to Q

A is the alphabet

Rank of letter C in alphabet is 3

$M \leftarrow 3 \text{ } 5p \text{ 'THREESHORTWORDS'}$
 M

A matrix of characters

THREE
SHORT
WORDS

$J \leftarrow A_1 M$
 J

Ranking of M produces a matrix

20 8 18 5 5
19 8 15 18 20
23 15 18 4 19
 $A[J]$

Indexing by a matrix produces
a matrix

THREE
SHORT
WORDS

3?5
5 1 2
6?5
DOMAIN ERROR

Random choice of 3 out of 5
without replacement

6?5

\wedge

$X \leftarrow 8?8$

A random permutation vector

X

4 6 7 2 5 1 8 3

$\uparrow X$

Grading of X

6 4 8 1 5 2 3 7

$X[\uparrow X]$

Arrange in ascending order

1 2 3 4 5 6 7 8

$X[\downarrow X]$

Arrange in descending order

8 7 6 5 4 3 2 1

$U \leftarrow A \in \text{'NOW IS THE TIME'}$

Membership

$\text{'01' } [1+U]$

00001001100011100011001000

U/A

EHIMNOSTW

$(18) \in 3 \text{ } 7 \text{ } 5$

0 0 1 0 1 0 1 0

Appendix B

ADVANCED EXAMPLES

This section presents a set of examples less elementary than those of Appendix A. These examples are all contained in Workspace *ADVANCEDEX* of Library 1. A user may therefore load, use, and trace any of the functions as an aid to understanding their behavior. Displays of intermediate results may also be inserted. For example, the statement

```
P←(P,0)+0,P
```

occurring in a function could be changed (perhaps by the use of line editing) to the following form:

```
□←P←(□←P,0)+□←0,P
```

Each execution of the statement will now perform as before, except that each of the results *0,P* and *P,0* and *P* will be typed out as well (in that sequence).

Programming techniques can be learned from a similar study of any well-written set of functions. All of the workspaces of library 1 may be used as a source of functions for such study.

The index origin in the workspace *ADVANCEDEX* is set to 1.

```
)LOAD 1 ADVANCEDEX
ADVANCEDEX SAVED 07/20/68 28.12.10
)FNS
AH      ASSOC  BIN      COMB      DTH      ENTER    F        FC
GC      GCD    GCV      HILB     HTD      IN       INV     INVLP
IN1     LFC    LOOKUP  PACK     PALL     PER      PERM    PO
POL     POLY   POLYB   RESET   TIME     TRUTH   UNPACK  ZERO
)VARS
DAH     DASSOC  DBIN     DCOMB    DDTH     DENTER   DESCRIBE
DF      DFC     DGC      DGCD     DGCV     DHILB   DHTD    DIN
DINV    DINVP   DIN1     DLFC     DLOOKUP  DPACK   DPALL   DPER
DPERM   DPO     DPOL     DPOLY   DPOLYB   DTIME   DTRUTH  DUNPACK
DZERO   TIMER
```

```
DESCRIBE
```

EACH OF THE VARIABLES OF THIS WORKSPACE WHICH BEGINS WITH THE LETTER D IS THE DESCRIPTION OF THE FUNCTION WHOSE NAME IS OBTAINED BY REMOVING THE D. FOR FURTHER DETAILS SEE APPENDIX B OF THE APL\360 USER'S MANUAL.

DPACK

THE FUNCTIONS PACK AND UNPACK ILLUSTRATE THE USE OF THE ENCODE AND DECODE FUNCTIONS IN TRANSFORMING BETWEEN A FOUR-NUMBER ENCODING OF SERIAL NUMBER (1 TO 9999), MONTH, DAY, AND YEAR, AND A SINGLE-NUMBER ENCODING OF THE SAME DATA.

```
    ∇PACK[□]∇
  ∇ Z←PACK X
[1] Z← 10000 12 31 100 1X-1
  ∇
```

```
    ∇UNPACK[□]∇
  ∇ Z←UNPACK X
[1] Z←1+ 10000 12 31 100 TX
  ∇
```

```
    P←PACK 2314 7 17 68
    P
86063867
    UNPACK P
2314 7 17 68
    UNPACK PACK 2311 9 21 72
2311 9 21 72
    PACK UNPACK 92137142
92137142
    PACK 1 1 31 1
3000
    UNPACK 3000
1 1 31 1
```

DENTER

THE FUNCTIONS ENTER, LOOKUP, AND RESET ILLUSTRATE A METHOD OF CONSTRUCTING AND USING LISTS OF VARIABLE LENGTH DATA, REPRESENTING EACH LIST BY A VECTOR OF CHARACTERS AND A VECTOR OF INDICES. ENTER AND LOOKUP EACH REQUEST INPUT (BY \square) UNTIL AN EMPTY VECTOR (CARRIAGE RETURN ALONE) IS ENTERED.

RESET RESETS LISTS (USE BEFORE ENTER AND LOOKUP).
 ENTER ACCEPTS SUCCESSIVE ITEMS OF NAMES AND DATA.
 LOOKUP PRINTS DATA ASSOCIATED WITH EACH NAME ENTERED.

∇ ENTER[\square] ∇ ∇ ENTER;X [1] 'ENTER NAME' [2] X \leftarrow , \square [3] $\rightarrow 0 \times 10 = \rho X$ [4] NAMES \leftarrow NAMES,X [5] P1 \leftarrow P1, ρ NAMES [6] 'ENTER DATA' [7] DATA \leftarrow DATA, \square [8] P2 \leftarrow P2, ρ DATA [9] ' ' [10] $\rightarrow 1$ ∇	∇ LOOKUP[\square] ∇ ∇ LOOKUP;X;J [1] '?' [2] X \leftarrow , \square [3] $\rightarrow 0 \times 10 = \rho X$ [4] J \leftarrow ((1 \div P1)- $\bar{1}$ \div P1)= ρX)/ $\bar{1}$ $\bar{1}$ \div ρ P1 [5] J \leftarrow (NAMES[P1[J] \circ .+ $\bar{1}$ ρ X] \wedge .=X)/J [6] \rightarrow (0 1 = ρ J)/ 10 8 [7] $\rightarrow 1$, ρ \square \leftarrow 'MORE THAN ONE SUCH NAME' [8] DATA[P2[J]+ $\bar{1}$ -/ ρ P2[1 0 +J]] [9] $\rightarrow 1$ [10] 'NO SUCH NAME' [11] $\rightarrow 1$ ∇
---	--

∇ RESET[\square] ∇
 ∇ RESET
 [1] NAMES \leftarrow DATA \leftarrow ρ P1 \leftarrow P2 \leftarrow 0
 ∇

RESET
 ENTER
 ENTER NAME
 J. ARMSTRONG
 ENTER DATA
 PRESIDENT

ENTER NAME
 H. LEVINE
 ENTER DATA
 VICE-PRESIDENT

ENTER NAME

LOOKUP
 ?
 H. LEVINE
 VICE-PRESIDENT
 ?
 L. YAVNER
 NO SUCH NAME
 ?

DIN

THE FUNCTIONS IN AND IN1 TAKE TWO ARGUMENTS; THE FIRST IS A WORD (I.E., A VECTOR) WHOSE OCCURRENCES IN THE SECOND ARGUMENT ARE TO BE DETERMINED. THE RESULT IS A VECTOR OF INDICES OF THE FIRST LETTER OF EACH OCCURRENCE. THE FUNCTION IN DETERMINES ALL OCCURRENCES, WHEREAS IN1 DETERMINES ONLY ALL NON-OVERLAPPING OCCURRENCES BY FIRST APPLYING THE FUNCTION IN AND THEN SUPPRESSING ALL OVERLAPS.

```

    ∇IN[ ]∇
    ∇ Z←A IN B;J
[ 1] J←(A[1]=B)/1ρB
[ 2] J←(J≤1+(ρB)-ρA)/J
[ 3] Z←(B[J°.+-1+1ρA]∧.=A)/J
    ∇
    ∇IN1[ ]∇
    ∇ T←A IN1 B
[ 1] T←A IN B
[ 2] →2×J<ρT←(∼(1ρT)∈J+1+((ρA)>|-[1](2,1+ρT)ρT)11)/T
    ∇

```

```

    W←'THE'
    T←'THE MEN THEN WENT HOME.'
    W IN T
1 9
    W IN1 T
1 9
    'ABA' IN 'NOWABABABABABABABA'
4 6 8 10 12 14 16
    'ABA' IN1 'NOWABABABABABABABA'
4 8 12 16

```

DTRUTH

THE FUNCTION TRUTH PRODUCES THE MATRIX OF ARGUMENTS OF THE TRUTH TABLE FOR N LOGICAL VARIABLES.

```

    ∇TRUTH[ ]∇
    ∇ Z←TRUTH N
[ 1] Z←2|L(1+12*N)°÷2*N-1N
    ∇
    TRUTH 3
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
    (TRUTH 3)+.×φ2*-1+13
0 1 2 3 4 5 6 7

```

DGCD

THE FUNCTIONS GCD AND GC EACH EMPLOY THE EUCLIDEAN ALGORITHM TO PRODUCE THE GREATEST COMMON DIVISOR. GCD EMPLOYS TWO SCALAR ARGUMENTS, WHEREAS GC EMPLOYS A SINGLE ARGUMENT WHICH IS EXPECTED TO BE A TWO-ELEMENT VECTOR.

THE FUNCTION GCV YIELDS THE GREATEST COMMON DIVISOR OF ALL ELEMENTS OF A VECTOR OF TWO OR MORE ELEMENTS.

```

      ∇GCD[ ]∇
    ∇ Z←M GCD N
[ 1]  Z←M
[ 2]  M←M|N
[ 3]  N←Z
[ 4]  →0≠M
    ∇
      ∇GC[ ]∇
    ∇ Z←GC M
[ 1]  →0≠1→M←ϕM[ 1],Z←|/M
    ∇
      ∇GCV[ ]∇
    ∇ Z←GCV W;A
[ 1]  →1≠ρW←Z,(A≠0)/A←(Z←|/W)|W
    ∇
```

```

      84 GCD 90
6
      90 GCD 84
6
      GC 90 84
6
      GCV 90 84
6
      GCV 90 84 105
3
```

DBIN

THE FUNCTION BIN PRODUCES ALL BINOMIAL COEFFICIENTS UP TO ORDER N

```

      ∇BIN[ ]∇
    ∇ Z←BIN N
[ 1]  Z←[ϕ(0,1N)∘.!0,1N]
    ∇
```

```

      BIN 4
1 0 0 0 0
1 1 0 0 0
1 2 1 0 0
1 3 3 1 0
1 4 6 4 1
```

DPOLY

THE FUNCTIONS POLY, POL, PO, AND POLYB EACH EVALUATE A POLYNOMIAL (OR POLYNOMIALS), WHOSE COEFFICIENTS ARE DETERMINED BY THE FIRST ARGUMENT, AND WHOSE POINT (OR POINTS) OF EVALUATION IS DETERMINED BY THE SECOND ARGUMENT. THE COEFFICIENTS ARE IN ASCENDING ORDER OF ASSOCIATED POWERS.

POLY SCALAR RIGHT ARGUMENT ONLY.
 POL SCALAR RIGHT ARGUMENT ONLY (USES INNER PRODUCT).
 POLYB SCALAR RIGHT ARGUMENT ONLY (USES BASE VALUE).
 PO APPLIES TO ARGUMENTS OF ANY RANK. THE VECTORS ALONG THE FIRST COORDINATE OF THE FIRST ARGUMENT ARE THE COEFFICIENTS OF THE POLYNOMIALS WHICH ARE EVALUATED FOR EACH ELEMENT OF THE SECOND ARGUMENT.

	∇ POLY[] ∇		∇ POLYB[] ∇
	∇ Z+C POLY X		∇ Z+C POLYB X
[1]	Z←+/C×X* ⁻¹ +1ρ,C	[1]	Z←X1φC
	∇		∇
	∇ POL[] ∇		∇ PO[] ∇
	∇ Z+C POL X		∇ Z+C PO X
[1]	Z←(X* ⁻¹ +1ρ,C)+.×C	[1]	Z←(X°. * ⁻¹ +11ρρC)+.×C
	∇		∇

C←1 2 3 4
 C POLYB 3
 142
 (C POLY 3)∧.= (C POLYB 3), (C POL 3), C PO 3
 1
 C PO 1 2 3 4 5 6
 10 49 142 313 586 985
 □←M←φBIN 5

1	1	1	1	1	1
0	1	2	3	4	5
0	0	1	3	6	10
0	0	0	1	4	10
0	0	0	0	1	5
0	0	0	0	0	1

{M PO 16

1	2	4	8	16	32
1	3	9	27	81	243
1	4	16	64	256	1024
1	5	25	125	625	3125
1	6	36	216	1296	7776
1	7	49	343	2401	16807

DTIME

THE FUNCTION TIME YIELDS THE AMOUNT (IN MINUTES, SECONDS, AND 60THS OF A SECOND) OF CPU TIME USED SINCE ITS LAST PREVIOUS EXECUTION. IT IS USEFUL IN MEASURING THE EXECUTION TIMES OF OTHER FUNCTIONS. THE VARIABLE 'TIMER' IS ASSIGNED THE VALUE OF THE CUMULATIVE CPU TIME AT EACH EXECUTION OF THE FUNCTION TIME.

```

    ∇ TIME[ ] ∇
    ∇ Z←TIME;T
[1] Z← 60 60 60 τ(T←I21)-TIMER
[2] TIMER←T
    ∇
    DCOMB

```

THE FUNCTION COMB EMPLOYS RECURSIVE DEFINITION TO PRODUCE A 2!N BY 2 MATRIX OF ALL POSSIBLE PAIRS OF ELEMENTS FROM 1N.

THE FUNCTION FC SHOWS AN ALTERNATE METHOD WHICH YIELDS THE SAME PAIRS BUT IN A DIFFERENT ORDER.

THE FUNCTION LFC EMPLOYS FC TO GENERATE LETTER PAIRS.

```

    ∇ COMB[ ] ∇
    ∇ C←COMB N;A;B
[1] →0×1N<2
[2] →0×1N=2×1ρC← 1 2 ρ 1 2
[3] A←COMB N-1
[4] C←((ρA)+(N-1),0)ρ(,A),,(1N-1)°. [0,N
    ∇
    ∇ FC[ ] ∇
    ∇ C←FC N;A;B
[1] B←(1N)°. +Nρ0
[2] A←(1N)°. +1N
[3] C←(2,N×N)ρ(,B),,A
[4] C←Q(C[2;]≤N)/C
    ∇
    ∇ LFC[ ] ∇
    ∇ Z←LFC N
[1] Z←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'[FC N]
    ∇

```

		<i>TIME</i>						
0	0	35						
		<i>TIME</i>						
0	0	2						
		COMB 4			FC 4			LFC 4
1	2		1	2				AB
1	3		1	3				AC
2	3		1	4				AD
1	4		2	3				BC
2	4		2	4				BD
3	4		3	4				CD
		<i>TIME</i>			<i>TIME</i>			<i>TIME</i>
0	0	12	0	0	8	0	0	7

	Z←COMB	15		Z←FC	15
	ρZ			ρZ	
105	2		105	2	
	TIME			TIME	
0	1	4	0	0	29

DDTH

THE FUNCTIONS DTH, HTD, AND AH CONCERN HEXADECIMAL NUMBERS LIMITED TO 8 DIGITS AND EMPLOYING THE CHARACTERS 0123456789ABCDEF. NEGATIVE NUMBERS ARE REPRESENTED IN 2'S COMPLEMENT FORM, WITH ANY OF THE CHARACTERS 8 THROUGH F IN THE LEFTMOST POSITION (OF EIGHT). LEADING ZEROS MAY BE OMITTED.

DTH CONVERTS DECIMAL TO HEXADECIMAL.
 HTD CONVERTS HEXADECIMAL TO DECIMAL.
 AH ADDS HEXADECIMAL NUMBERS.

```

    ∇DTH[□]∇
  ∇ R←DTH X
[1] R←,( '0123456789ABCDEF' ) [ 1+(8ρ16)⊔X ]
  ∇
    ∇HTD[□]∇
  ∇ R←HTD X
[1] R←((8-ρ,X)ρ'0'),X
[2] R←[(16⊔1+'0123456789ABCDEF'⊔R)-(2*32)×R[1]ε'89ABCDEF'
[3] →4×~^/Xε'0123456789ABCDEF'
[4] R←''
[5] 'NUMBER IS NOT HEX'
  ∇
    ∇AH[□]∇
  ∇ R←A AH B
[1] R←DTH(HTD A)+HTD B
  ∇
    Z←DTH 1776
    Z
000006F0
    HTD Z
1776
    Z AH Z
00000DE0
    HTD Z AH Z
3552
    HTD '000006F0'
1776
    HTD '90000000'
-1879048192
    HTD '00049HFG'
NUMBER IS NOT HEX

```

DZERO

THE FUNCTION ZERO EMPLOYS THE METHOD OF FALSE POSITION TO DETERMINE TO WITHIN A TOLERANCE TOL A ROOT OF THE FUNCTION F LYING BETWEEN THE BOUNDS B[1] AND B[2]. IT IS ASSUMED THAT F B[1] AND F B[2] ARE OF OPPOSITE SIGN. THE FUNCTION F IS A SPECIFIC POLYNOMIAL, BUT CAN BE CHANGED TO ANY DESIRED FUNCTION.

```

      VZERO[ ]V
      V Z←TOL ZERO B;T
[1]   →0×TOL≥|T←F Z←0.5×+/B
[2]   →1,B[2] (0<T)≠0<F B]←Z
      V

      VF[ ]V
      V Z←F X
[1]   Z← -20 18 3 -5 1 PO X
      V

      X←-4+19
-3   -2 -1 0 1 2 3 4 5
      F X
169 12 -29 -20 -3 4 7 36 145
      TIME
0 1 19
      R←1E-6 ZERO -2 -1
-1.845121413
      TIME
0 2 36
      F R
7.14140814E-7
      TIME
0 0 2
      F R←1E-10 ZERO 1 2
1.26397094
-1.813305062E-11
      TIME
0 3 46
      F R←1E-6 ZERO 1 2
1.263970852
-8.51888359E-7
      TIME
0 2 13

```

DHILB

THE FUNCTION HILB PRODUCES A HILBERT MATRIX OF ORDER N.

```

    ∇ HILB[ ] ∇
    ∇ Z ← HILB N
[1]  Z ← ÷-1 + (1N) ° . + 1N
    ∇
    HILB 3

1          0.5          0.3333333333
0.5        0.3333333333  0.25
0.3333333333  0.25      0.2

```

DINV

THE FUNCTIONS INV AND INVP EACH PRODUCE THE INVERSE OF THE MATRIX ARGUMENT SUPPLIED, EMPLOYING GAUSS-JORDAN (I.E., COMPLETE) ELIMINATION. INVP EMPLOYS PIVOTING AND INV DOES NOT.

THE FIRST LINE APPENDS THE UNIT VECTOR 1 ≥ 1N AS THE LAST COLUMN OF THE ARGUMENT AND THE SECOND LINE (LINE 4 IN INVP) PERFORMS AT EACH ITERATION ONE OF THE N COMPLETE INVERSIONS REQUIRED. SEE EXERCISE 1.40 OF IVERSON, A PROGRAMMING LANGUAGE, WILEY, 1962.

```

    ∇ INV[ ] ∇
    ∇ Z ← INV M; I; J
[1]  M ← ϕ(1 0 + ρM) ρ(, ϕM), ~J+1 < 1I+1 ↑ ρM
[2]  M ← 1 ϕ(J, 1) ϕ[1] M - (J × M[; 1]) ° . × M[1; ] ← M[1; ] ÷ M[1; 1]
[3]  → 2 × 10 ≠ I+I-1
[4]  Z ← M[; 1 ↑ ρM]
    ∇
    ∇ INVP[ ] ∇
    ∇ Z ← INVP M; I; J; K; P
[1]  M ← ϕ(1 0 + ρM) ρ(, ϕM), ~J+1 < P+1I+1 ↑ ρM
[2]  M[K, 1; 1 ρP] ← M[1, K+( |M[ 1I; 1 ]) 1 ↑ / |M[ 1I; 1 ]; 1 ρP]
[3]  P ← 1 ϕ P, 0 ρ P[K, 1] ← P[1, K]
[4]  M ← 1 ϕ(J, 1) ϕ[1] M - (J × M[; 1]) ° . × M[1; ] ← M[1; ] ÷ M[1; 1]
[5]  → 2 × 10 ≠ I+I-1
[6]  Z ← M[; 1 P]
    ∇
    □ ← N ← INV M ← HILB 3

```

```

    9          -36          30
-36          192          -180
30          -180          180
M+ . × N

```

```

1.000000000E0    2.842170943E-14    -6.039613254E-14
1.421085472E-14  1.000000000E0    -1.065814104E-14
4.662936703E-15  3.197442311E-14    1.000000000E0

```

DPALL

THE FUNCTION PALL PRODUCES THE MATRIX OF ALL PERMUTATIONS OF ORDER N. THE FUNCTION PERM WHICH IT USES PRODUCES THE B-TH PERMUTATION OF ORDER N BY A METHOD DUE TO L.J. WOODRUM.

THE FUNCTION PER EMPLOYS RECURSIVE DEFINITION, AND PRODUCES ALL PERMUTATIONS BY A METHOD MUCH FASTER THAN THAT USED IN THE FUNCTION PALL. THE PERMUTATIONS ARE PRODUCED IN THE OPPOSITE ORDER.

```

    ∇PALL[□]∇
    ∇ Z←PALL N;I
[1] Z←(!N),N)ρ0
[2] I←1
[3] Z[I;]←N PERM I
[4] →3×(!N)≥I←I+1
    ∇
    ∇PERM[□]∇
    ∇ Z←A PERM B;I;Y
[1] I←ρZ←1+(Φ1A)†B-1
[2] →0×10=I←I-1
[3] Z[Y]←Z[Y]+Z[I]≤Z[Y←I+1A-I]
[4] →2
    ∇
    ∇PER[□]∇
    ∇ P←PER M;X;Y;Z
[1] →0×1M=P← 1 1 ρ1
[2] Z←PER M-1
[3] P←1X←0
[4] →0×1M<X←X+1
[5] Y←(~(1M)∈X)\Z
[6] Y[;X]←M
[7] P←((X×!M-1),M)ρ(,P),,Y
[8] →4
    ∇
    PALL 3

    1 2 3
    1 3 2
    2 1 3
    2 3 1
    3 1 2
    3 2 1
    TIME
0 3 7
    Z←PALL 3
    TIME
0 0 49
    Z←PALL 5
    TIME
0 25 10
    Z←PER 5
    TIME
0 1 12

```

DASSOC

THE FUNCTION ASSOC TESTS ANY PUTATIVE GROUP MULTIPLICATION TABLE M (ASSUMING GROUP ELEMENTS 1 1 ρ ρ M) FOR ASSOCIATIVITY AND YIELDS A VALUE 1 IF IT IS ASSOCIATIVE, 0 OTHERWISE.

```

    ∇ ASSOC [ ] ∇
  ∇ Z ← ASSOC M
[1] Z ← ^ / , M [ M ; ] = M [ ; M ]
  ∇

```

```

M ← (15) ϕ 5 5 ρ 15
M

```

```

2 3 4 5 1
3 4 5 1 2
4 5 1 2 3
5 1 2 3 4
1 2 3 4 5

```

```

    TIME
0 0 13
    ASSOC M

```

```

1
    TIME
0 0 9
M ← 0 0 1 0 0 ϕ M
M

```

```

2 3 4 5 1
3 4 5 1 2
5 1 2 3 4
5 1 2 3 4
1 2 3 4 5

```

```

    ASSOC M
0

```

```

    TIME
0 0 10
M ← ? 10 10 ρ 10
ρ M

```

```

10 10
    TIME
0 0 3
    ASSOC M

```

```

0
    TIME
0 0 45

```

)LOAD 1 NEWS
SAVED 10.44.08 07/12/68

DESCRIBE

THIS WORKSPACE PROVIDES INFORMATION ABOUT THE OPERATION AND USE OF APL. THE FUNCTIONS OF INTEREST TO THE USER ARE APLNOW, INDEX, PRINT, AND SCHEDULE.

APLNOW TAKES AS ITS SINGLE ARGUMENT A THREE-ELEMENT VECTOR REPRESENTING A DATE, AS MONTH, DAY, YEAR. APLNOW PRINTS NOTES ON THE STATUS OF THE APL SYSTEM; FOR INSTANCE, RECENTLY ADDED FEATURES, TEMPORARY RESTRICTIONS, OR ADVICE ON PROGRAMMING OR TERMINAL OPERATION. ONLY THOSE NOTES ENTERED INTO APLNOW ON OR AFTER THE DATE GIVEN AS AN ARGUMENT ARE PRINTED.

INDEX TAKES NO ARGUMENT. IT PRINTS INDICES, DATES, AND THE FIRST FEW WORDS OF EACH NOTE IN APLNOW.

PRINT TAKES AS ITS SINGLE ARGUMENT THE INDEX (AS INDICATED BY THE INDEX FUNCTION) OF A NOTE FROM APLNOW, AND PRINTS THE NOTE.

SCHEDULE TAKES NO ARGUMENT. IT INDICATES THE REGULAR DAILY APL SCHEDULE, AND ALL ANTICIPATED DEVIATIONS FROM THE NORMAL SCHEDULE.

BIBLIOGRAPHY

- Berry, P.C., APL\360 Primer, IBM Corporation, 1968.
- Berry, P.C., APL\1130 Primer, IBM Corporation, 1968.
- Breed, L.M., and R.H. Lathwell, "The Implementation of APL\360", ACM Symposium on Experimental Systems for Applied Mathematics, Academic Press, 1968.
- Falkoff, A.D., and K.E. Iverson, "The APL\360 Terminal System", ACM Symposium on Experimental Systems for Applied Mathematics, Academic Press, 1968.
- Falkoff, A.D., K.E. Iverson, and E.H. Sussenguth, "A Formal Description of System/360", IBM Systems Journal, Volume 3, Number 3, 1964.
- Iverson, K.E., A Programming Language, Wiley, 1962.
- Iverson, K.E., Elementary Functions: an algorithmic treatment, Science Research Associates, 1966.
- Iverson, K.E., "The Role of Computers in Teaching", Queen's Papers in Pure and Applied Mathematics, Volume 13, Queen's University, Kingston, Canada, 1968.
- Lathwell, R.H., APL\360: Operator's Manual, IBM Corporation, 1968.
- Lathwell, R.H., APL\360: System Generation and Library Maintenance, IBM Corporation, 1968.
- Pakin, S., APL 360 Reference Manual, Science Research Associates, 1968.
- Rose, A.J., Videotaped APL Course, IBM Corporation, 1968.
- Smillie, K.W., Statpack 1: An APL Statistical Package, Publication No. 9, Department of Computing Science, University of Alberta, Edmonton, Canada, 1968.

INDEX

Absolute value
 see Magnitude
 Account number 1.8 2.2-3 2.23
 2.28
 Accounting information 1.11 2.10
ALREADY SIGNED ON 1.9 2.7
 And 3.5 3.7 3.9 3.34
 Announcements, public address 1.9
 2.32
 APL
 character set 1.3
 exercise 4.5-6
 information B.13
 operations 2.1-3 2.5
 Application packages 2.13
 Arccos 3.7
 Arccosh 3.7
 Arcsin 3.7
 Arcsinh 3.7
 Arctan 3.7 A.2
 Arctanh 3.7 A.2
 Arguments 3.5 3.8-9 3.11 3.14-16
 3.28 3.37 3.41-44
 left 3.8
 right 3.3
 Array 3.26-33 A.6-8
 extend scalar functions to 3.5
 3.33
 functions on 2.20 3.33-36 4.3
 A.8-10
 Arrow 3.1-5 3.21 3.24
 Associativity test B.12
 Asterisk 2.30 3.21
 Attention 1.7 2.4 3.1 3.18

 Backspace 1.7 3.3
 Base value 3.42 A.13
 Beta function 3.9
 Binomial coefficient 3.7 A.5 B.5
 Body of defined function 3.10-11
 A.3
 Bounce 2.8-9 2.11
 Brackets 3.10 3.17
 Branch 3.1 3.11 3.20-21 3.45
 A.3
 example 3.12
 Business accounting 2.13
 Byte 3.46-47

 Caret 3.3
 Catenation 3.27 3.29 3.38 4.2
 Ceiling 3.7-8 A.2
 Character
 error 3.4
 input 3.24-25
 output 3.26 3.33
 Characteristic 3.43
 Circle function 3.34
 Circular functions 3.7
CLEAR 2.7 2.14 3.33
CLEAR WS 2.6 2.15
 Column 3.30-32 3.35 3.42 A.7
 Column catenator 4.2
 Colon 2.2-3 2.10 2.14 3.17
 Combinations B.7
 Comma 3.27-28
 Commands 2.5-7
 table 2.7
 Comment 3.45
 Communication commands 2.5 2.32-33
 action and notes 2.33-35
 table 2.7
 Comparison 3.8 A.9
 Complete beta function 3.9
 Compound character 3.3
 Compound expression 3.3 3.14
 Compress 3.38 3.41-42 A.13
 Computer
 access 1.1
 failure 2.8
 time used 1.11 3.47
 Conformability 3.33 3.36
 Connection
 broken 1.6 2.4 2.8 2.10
 computer 1.1
 establish 1.4-6
 hold 2.11-12
 Constants 3.2
 CONTINUE 1.11 2.7-9 2.11-12
 Continuity of work 2.22
 Coordinates of array 3.32 3.35
 3.37
 COPY 2.7 2.13-14 3.33
 action and notes 2.15-18
 Correction 1.7 A.1 A.4
 Cosh 3.7
 Cosine 3.7 A.2

 Data button 1.6
 Date 3.47
 Date response 2.9-10
 Deal 3.7 3.38 3.45
 Decimal form 3.2 4.3
 Decode 3.38 3.42 B.2
 Defined function 2.3 3.1-16 A.3
 example 3.11 A.3
 explicit argument and result 3.14
 3.15
 list names of 2.28
 start and end 3.10
 table of forms 3.15
 use 3.16
 variables made local to 3.13
 Definition mode 2.5 3.17
 end 3.18 A.3
 reestablish 3.19
DEFN ERROR 3.4
 Del 3.10 3.17-19
 Del tilde 3.19
 Delete 3.17-19
DEPTH ERROR 3.4
 Diagonal 3.39 A.11
DIGITS, output maximum 2.7 2.21
 4.7
 Dimension 3.5 3.26-29 3.35
 Dimension and rank table 3.27
 Disconnect 1.10 2.8 2.11
 Display 3.17-18 A.4
 Divide 3.7 3.34
DOMAIN ERROR 3.4 3.44 4.3
 Drop 3.38 3.44
DROP 2.7 2.22 2.27
 Dyadic functions 3.7-8 3.34 3.36
 Dyadic circle function table 3.7

e
 see Exponential

Edit
 in function definition 3.18-19
 A.5
 text 2.13

Element-by-element 3.5 3.9 3.33
 A.1

Empty array 3.29 3.33
 Empty vector 3.10 3.29 A.2
 Encode 3.38 3.43 B.2
 Entries 1.6-7 3.2 3.24

Epsilon
 see Characteristic

Equal 3.7-9 3.34

Equipment 1.1 1.2

Erase 1.7 2.27 3.3
ERASE 2.7 2.13 2.20 3.19 A.5

Error 1.7
 Error report 3.3 3.5
 during defined function 3.5
 table 3.4

Evaluated input 3.24

Evaluation, order of
 see Order of execution

Exclusive or 3.9

Execution mode 2.5 3.19

Execution, order of
 see Order of execution

Expand 3.29 3.38 3.42 A.13

Exponential 3.7
 Exponential form 3.2 4.2-3 A.1
 Expression, compound 3.3 3.14

Factorial 3.7 3.9 3.16 A.3

Floor 3.7-8
FNS 2.7 2.28 3.22

Function name 2.19 3.2 3.11
 3.23

Function definition 3.10-11
 3.17-19 A.4-6

Functions
 see Circular, Defined, Dyadic,
 Halted, Library, Locked, Mixed,
 Monadic, Pendent, Primitive,
 Recursive definition, Scalar,
 Suspended, System dependent

Fuzz 3.8

Gamma function 3.9

Gauss-Jordan elimination B.10

Global object 2.3 2.15-20 3.23

Global variable 2.13-14 2.29 3.13
 A.3

Grade up and down 3.38 3.44

Graph 4.2 4.5

Greater 3.7 3.9

Greatest common divisor A.4 B.5
GROUP 2.7 2.13-14 2.19 2.29
GRP, GRPS 2.7 2.29-30 3.22

Halted function 2.3 2.22 2.30-31

Header 3.13 3.15-19 A.3

Hexadecimal number B.8

Homonyms 2.3 3.22-23

Hyperbolic functions
 see Circular functions

I-beam functions 3.20 3.46-48

Identity element table 3.34

IMPROPER LIBRARY REFERENCE 2.7
 2.26-27 2.31

INCORRECT COMMAND 2.6-7

INCORRECT SIGN-ON 1.9 2.7

Index 3.26 3.38

INDEX ERROR 3.4 3.30 3.34

Index generator 3.9 3.38 A.2

Indexing 3.26 3.30-33 A.14-15

Index of 3.38 3.43

Index origin 2.20 3.32-34 A.8

Inner product 3.5 3.33 3.35 A.9
 table 3.36

Input
 character 3.25
 escape from loop 3.25
 evaluated 3.24
 position 1.8 2.4

Input and output 3.23-26 A.6

Inquiry command 1.8 2.5 2.28
 action and notes 2.28-32
 table 2.7

Insert by backspacing 3.3

Interrupt 1.1 2.4 2.9 3.20

Iota
 see Index generator, Index of

Iteration counter 3.13

Key 1.8 2.3 2.14

Keyboard 1.3
 locked 1.6-8 2.32-33
 unlocked 1.10 2.4 2.34 3.24

Keying time 3.47

Label 3.17

LABEL ERROR 3.4

LENGTH ERROR 3.4 3.33 4.3

Less 3.7 3.9

LIB 2.7 2.31

Library 2.2
 see Private, Public

LIBRARY 1 4.1
ADVANCEDDEX B.1-12
APLCOURSE 4.5
NEWS B.13
TYPEDRILL 4.8-9
WSFNS 3.32 4.7

Library control command 2.5
 2.22-24
 action and notes 2.25-27
 table 2.7

Library function 2.5 4.1-9

Library number 2.14 2.23

LIBRARY TABLE FULL 2.9

Line width 2.21

List, construct and use B.3

LOAD 2.7 2.22

Local variable 2.3 3.13 3.22

Lock 2.3 2.10

Locked function 3.5 3.9

Locked keyboard 1.6-8 2.4 2.32-33

Logarithm 3.7

Logical function 3.5 3.9

Logical result A.2

Magnitude 3.7-8
 Material implication 3.9
 Mathematical statements 3.1
 Matrix 3.26-27 3.33 3.35 A.7
 A.15
 Hilbert B.10
 Maximum 3.7 3.34 A.9
 Membership 2.30 3.38 3.43 A.15
 Message 1.8 2.21 2.32-33
 MESSAGE LOST 2.7 2.33
 Minimum 3.7 3.34
 Minus 3.2 3.7 3.34
 Mixed functions 3.37-45
 names 3.5
 sample A.10-15
 table 3.38-39
 Monadic functions 3.7-8
 Monadic transpose 3.40
 MSG, MSGN 2.7 2.33-34

 Names 2.2-3 2.14 2.19 2.23
 3.10 3.13
 active and latent referents 3.22
 3.23
 to obtain lists of 2.28-31
 Nand 3.7 3.9 3.34
 Natural logarithm 3.7
 Negation function 3.2
 Negative number 3.2 3.7 A.1
 NO SPACE 2.9
 Nor 3.7 3.9 3.34
 Not 3.7
 Not equal 3.7-9 3.33
 Not greater 3.7 3.9 3.34
 NOT GROUPED, NAME IN USE 2.7 2.19
 Not less 3.7 3.9 3.34
 NOT SAVED, THIS WS IS wsid 2.7
 2.27
 NOT SAVED, WS QUOTA USED UP 2.7
 2.27
 NOT WITH OPEN DEFINITION 2.7 2.16
 2.25 3.19
 NUMBER IN USE 1.9 2.7
 NUMBER LOCKED OUT 1.9 2.7
 NUMBER NOT IN SYSTEM 1.9 2.7
 Number of terminals connected 3.47
 Numbers 3.2 3.5 3.26

 OBJECT NOT FOUND 2.7 2.16
 Occurrences B.4
 OFF 2.7 2.10-11
 Operator 1.4-5 1.8
 OPR, OPRN 1.8 2.7 2.34-35
 Or 3.5 3.7 3.9 3.34
 Order of execution 3.45
 compound expression 3.3 A.1
 defined function 3.11
 ORIGIN 1.7 2.7 2.20 3.32 4.7
 Out of 3.34
 Outer product 3.5 3.33 3.35-36
 Output 2.1 2.21 3.23 3.25
 array 3.33
 fixed point 4.2
 heterogeneous 3.26
 maximum digits 2.21
 Overstrike 3.3-4 3.19 3.25
 3.35 3.41 3.45

 Parentheses in expression 3.3
 Parenthesis, right 1.8 2.5-6
 Password 1.8 2.3 2.10
 PCOPY 2.13 2.18
 Pendent function 2.3 2.30 3.21
 Permutation 3.32 3.37 A.8 A.14
 B.11
 Pi times 3.7
 Plus 3.7 3.34
 Polynomial B.6
 Ports 1.1
 PORTS 2.7 2.32
 Power 3.7 3.34
 Precision of numbers 2.21 3.5
 Primitive functions 3.1 3.7 3.14
 3.38
 names 3.5
 see Mixed, Scalar
 Printing element 1.1-3
 Private library 2.2-3 2.23
 Programming examples B.1-12
 Protecting copy command 2.13 2.18
 Public library 2.2 2.23 4.1

 Quad 3.24-25 3.47
 Quotation mark 3.2
 Quote quad 3.25 3.47

 Radices 3.42
 Random 3.7 3.45 A.15
 RANK ERROR 3.4 3.33 4.3
 Rank of array 3.26-27 A.14
 Ravel 3.27-28 3.38
 Reciprocal 3.7
 Recording terminal 1.4 2.8 2.24
 2.32 2.34-35
 Recursive definition 3.16 B.7
 B.11
 Reduction 3.5 3.33-35 A.9
 Representation 3.43 A.13
 Request button on 1050 terminal 1.8
 RESEND 1.7 3.4
 Reshape 3.28 3.38 A.7
 Residue 3.7-8 3.34
 Response 1.9-11 2.6 2.10 A.1
 table 2.7
 Results 3.2 3.14-15
 Return 1.6-7 3.3
 Reverse 3.38 3.41 A.12
 Revise 3.17
 Rho
 see Reshape, Size
 Roll 3.7
 Rotate 3.38 3.40 A.12
 Row 3.30-32 3.35 3.42 A.7

 SAVE 2.7 2.22 2.25-26
 Scalar 3.27 3.37
 Scalar functions 3.5-10
 extend to arrays 3.33-36
 sample A.2
 table 3.7
 Security 2.3
 Semicolon 3.13 3.26

SI, SIV 2.7 2.30-31 3.20-22
 Sign on 1.8-10 2.7
 Signum 3.7 A.3
 Sine 3.7 A.2
 Sine, cosine, tangent table A.11
 Sinh 3.7
 Size 3.5 3.27 3.38
 Spaces 2.6 3.2
 Specification 3.1
 multiple 3.45
 State indicator 2.1 2.22 3.20-21
 3.48 A.5
 with local variables 3.22
 Statement 3.1
 end of 3.3
 invalid 3.3 3.5 A.2
 number 3.10-11 3.17-18 3.48
 Statistical calculations 2.13
 Stop control 3.19 3.21
 Storage 2.1 2.24 3.46
 Store workspace 2.25-26
 Suspended function 2.30 3.20-21
 3.48
 Symbols 3.5 3.7 3.24-25 3.38
 3.45
SYMBOL TABLE FULL 3.4
SYNTAX ERROR 3.4
 System command 2.1 2.5-6
 during function definition 3.19
 table 2.7
 System dependent function 2.5 2.30
 3.46-48
 table 3.47
SYSTEM ERROR 3.4
 System information 1.9 3.46-48

 Tabular output 4.3
 Take 3.38 3.44
 Tangent 3.7 A.2
 Tanh 3.7
 Teaching 2.13
 Telephone number 1.5
 Terminal
 table of features and options 1.2
 1050 1.1 1.5 1.7-8
 2740 1.4-6
 2740-1 1.1
 2741 1.1 1.4-6 2.4
 Terminal control command 2.5 2.8-9
 action and notes 2.9-12
 table 2.7
 Terminal, sample session at A.1-15
 Terminal modes 2.5
 Time 1.9 1.11 2.9 3.46-48 B.7
 Timer on 1050 terminal 1.8
 Times 3.7 3.33
 Time, student response 3.47

 Trace 3.16 3.19 3.21 A.3
 Transmission signal 1.6-7
 Transpose 3.37-41 A.10-11
 Transposition table 3.40
 Trigonometric functions
 see Circular functions
 Trouble report 1.7 2.6
 table 2.7
 Typewriter entry 1.6-7 3.2 3.24
 A.1

 User code 1.11 2.28

 Value 3.11 3.14-15 3.17 3.25
 3.45
VALUE ERROR 3.4 3.15
 Variables
 local and global 2.3 3.13 3.22
 names 2.1-2 2.19 3.22-23
 value 3.1 3.14 3.32
VARS 2.7 2.29 3.22
 Vectors 3.9-10 3.26-27 3.35 A.1
 mixed functions apply 3.37
 numerical and character constant
 3.2 A.1
 scalar functions apply 3.9

WIDTH 2.7 2.21 3.25 4.7
 Work session
 forced end 2.8
 to end 1.10-11 2.10-12
 to start 1.8
 Workspace 1.9 2.1-2
 activate 2.14-15
 active 2.13 2.28
 capacity 2.1
 clear 1.10 2.6 2.14
 continue 1.10 2.8 2.24
 identification 2.14 2.23 2.31
 identification change 2.22-23
 index origin 3.33
 information transfer between 2.13
 in libraries 4.1
 name 2.2-3
 purge an active 2.24
 replace stored with active 2.25
 stored 2.2-3 2.23
 Workspace control command 2.5
 2.13-14
 action and notes 2.15-22
WS FULL 2.7 2.16 3.4
WSID 2.7 2.31
WS LOCKED 2.7 2.15
WS NOT FOUND 2.7 2.15 2.27

 Zero of a function B.9