DISCLAIMER

This program and its documentation have been contributed to the Program Information Department by an IBM employee and are provided by the IBM Corporation as part of its service to customers. The program and its documentation are essentially in the author's original form and have not been subjected to any formal testing. IBM makes no warranty expressed or implied as to the documentation, function, or performance of this program and the user of the program is expected to make the final evaluation as to the usefulness of the program in his own environment. There is no committed maintenance for the program.

Questions concerning the use of the program should be directed to the author or other designated party. Any changes to the program will be announced in the appropriate Catalog of Programs; however, the changes will not be distributed automatically to users. When such an announcement occurs, users should order only the material (documentation, machine readable or both) as indicated in the appropriate Catalog of Programs.
APL is a conversational implementation of the Iverson notation, an extremely concise mathematical notation with simple but rigorous syntax. This concise attribute virtually eliminates the -program- step in the problem-solving chain of problem--algorithm--program--solution. All operators of the notation, editing capabilities, and the capability to save and retrieve work spaces are provided. The implementation allows data to be structured as scalars, vectors, and matrices with up to 255 elements in any dimension. Numerical values are accurate to six decimal digits, and identifiers are up to 6 alphabetic characters. Input may come from the console typewriter, card reader or a typewriter terminal. The program is independent of the IBM monitor and requires a dedicated disk cartridge. Facilities are provided to generate the system, assign and delete workspaces, and dump/restore individual workspaces and their functions to cards.
Minimum configuration is 1131-2B and 1442 or 2501. A 2741 terminal and requisite RPQ are highly desirable. APL/1130 is written in 1130 Assembler Language.

TABLE OF CONTENTS

Program Contribution Form 1
Deck and Tape Keys 3
User Information Key 4
Preface 5
Operating Instructions 10
  GENERATING THE SYSTEM 10
  INITIAL PROGRAM LOAD 10
  USE OF CONSOLE SWITCHES 11
  SYSTEM MAINTENANCE COMMANDS 12
  SAMPLE TERMINAL OUTPUT 13.1
  APL SYSTEM LOAD 13.2


Authors: A. D. Falkoff
         K. E. Iverson

(Contains its own table of contents)
DECK KEY

For 1442 loading:

Load card:          APL\TR

TPL sector deck:    First card unsequenced

Next 8 cards 1442LD00-1442LD07

Next 5 cards IPL00000-IPL00004

APL system deck:    First card unsequenced

Next 8 cards 1442LD00-1442LD07

Next 438 cards IPL00000-IPL00437

Empty directories deck:  First card unsequenced

Next 8 cards 1442LD00-1442LD07

Next 9 cards DIR00000-DIR00008

For 2501 loading:

Identical to the decks above, except that the first 9 cards of

each deck are to be removed and replaced by:

First card unsequenced

Next 14 cards 2501LD00-2501LD13

(Three copies of this deck are included for user insertion- see

OPERATORY INSTRUCTIONS page 13,2)

TOTAL NUMBER OF CARDS 526

TAPE KEY (OPTIONAL MATERIAL)

1. The tape supplied as optional material, contains card images, blocked

   20 cards per block. The tape is unlabeled. There is 1 file on the

   tape, preceded by a single tape mark. Two tape marks follow the

   end of the file.

2. Data stored includes a complete 1130 job stream (including all

   required control cards necessary to assemble the system under

   Monitor II).

3. The tape was created by the System 360 DESE program. It may be

   'churned' back to cards by the same program or any appropriate

   tape to card utility program.

   * OPTIONAL MATERIAL WILL BE FORWARDED ONLY WHEN

   SPECIFICALLY REQUESTED.

USER INFORMATION KEY

The following items of interest may be found where indicated:

Purpose: See Preface

Advantages: See Preface

Restrictions and range: See page 56

Precision: See page 56

Program requirements: See Operating Instructions

System configuration: See page 20

Timing: Response to trivial operations is within 2 seconds. Other

responses are proportional to the amount of computation required.

Program modification aids: None provided.

Input and output description: See page 24 and page 45

Sample problem: See Appendix A, page 93
PREFACE

APL/1130 is a single-user implementation on a smaller machine of the APL/360 conversational terminal system that has been operating within IBM since the Fall of 1966. Conceived as an experimental system for exploring certain aspects of computer science, its purpose required that it operate under a realistic load in an environment that was not artificially constrained. To this end, the members of the IBM Research staff, and others, were encouraged to learn the APL language and use the APL system. It was not anticipated that it would have the impact that it did:

In twenty months of regularly scheduled operation, clocking well over 100,000 terminal-hours of use, the availability of APL/360 has materially changed the computing habits of the Research organization.

Heavy users of batch processing have turned to APL for on-line development of their algorithms.

Many laboratory data reduction chores formerly done by batch operation or desk calculator are now executed in a timely way at local terminals, using locally written, stored APL programs.

Automatic collection of experimental data has, in many instances, been put on-line to APL.

Routine correspondence and technical papers are prepared at terminals, with the help of text-handling programs written in APL.

Many professional staff members who formerly were indifferent to, or actively resisted, the use of computers, have become steady users of the APL system.

In addition to IBM Research personnel, use of the system was offered to other locations within the IBM Company, and it was also used experimentally in elementary and secondary schools and in universities. The general findings may be summarized as follows:

Although APL is easy for a beginner to learn, non-programmers (as well as programmers) often develop an interest in sophisticated use of the language, because of its analytical power and mathematical structure.

The primitive array operations of APL make it a good language for scientific problems, text handling, and general data processing, because arrays are fundamental to all of these applications.

The use of a powerful, readily accessible computational facility can materially change the quality and orientation of an academic course.

Other things being equal, acceptance of conversational computing as a general mode of operation is strongly dependent upon its reliability and availability -- regularly scheduled hours are essential, and the more the better, including nights and weekends.

APL terminal systems are characterized by the following:
Simple, uniform rules of syntax
Use of common symbols for the ordinary arithmetic operations
Free-form decimal input
A large set of primitive operators
Use of defined functions (programs) with the same facility and syntactic variety as primitive operators
Fast response
A library structure built around workspaces that hold both programs and data
An immediate-execution mode completely free of irrelevant keywords
A comprehensive, integrated set of system commands for managing workspaces and libraries, and for other essential functions
Three levels of security; account numbers, workspaces, and programs can be individually locked against use or display
Visual fidelity between hard copy and transmitted entries, which ensures reproducibility of results
Succinct diagnostic reports
New release. This is the second release of APL\ll130. It embodies a number of new features which are described in detail in the APL\ll130 User's Manual. They include:

1. Six-character identifiers.
2. Labels.
3. Extended function editing.
5. Provision for connecting a remote 2741 terminal.
6. A dynamic method of localizing names as described in the section on Hominonyms.
7. A new command )SIV to display the state indicator together with the list of local variables for each halted function.
8. A system of locks and keys for security of account numbers and workspaces.
9. The signum function (denoted by \texttt{*}) has been added.
10. The following changes in notation have been made:

<table>
<thead>
<tr>
<th>New notation</th>
<th>Replaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>)ERASE F</td>
<td>IF</td>
</tr>
<tr>
<td>葑</td>
<td>a</td>
</tr>
<tr>
<td>TAP</td>
<td>LF</td>
</tr>
</tbody>
</table>

11. A set of COPY commands permits the copying of all or part of one workspace into another.

Instructional use. APL\ll130 is well suited for use by secondary schools for instructional purposes. In immediate execution mode, APL\ll130 executes each line as the student types it and replies before he types the following line. This give-and-take between student and computer enhances student involvement and augments the educational process. Function definition mode enables a student to build and store functions for later use which become a framework for the subject matter.

APL\ll130 is suitable not only for student problem-solving in science and mathematics courses, but also for drill and exercise in a programmed-teaching mode. The system provides a formal method of function definition and a method of saving data and programs from session to session. APL also facilitates individual student experimentation; the student may draw from his own experience and library of functions or from a library of commonly used programs maintained by an instructor.

Under normal circumstances, it is difficult for a teacher to be able to provide adequate instruction for all the levels of student achievement represented in one classroom. APL\ll130 can help accommodate the individual differences present in several ways. First, the computer system can provide drill and exercise for those students who require more practice in problems than the instructor can provide. Secondly, APL\ll130 provides unlimited opportunity for individual experimentation and exploration for the above-average student; a gifted child no longer need be inhibited by the requirements of his classmates. Finally, APL\ll130 can be used by teachers to prepare numerical examples which would otherwise require tedious time-consuming hand calculations.

In addition, the following features of APL\ll130 are of particular significance in instructional use:

1. Since APL is simple and concise and is closely related to the ordinary notation of algebra, a student's attention can be devoted almost entirely to solving the problem, rather than being diverted by irrelevant details of a computer.
2. Basic APL operators are single characters. Economy of symbols is gained by using each symbol as both a monadic and dyadic operator, just as a minus sign is used in ordinary arithmetic.
3. APL provides standard operators to perform common functions which in other languages must be programmed. For example, polynomial evaluation, random number generation, summation, matrix multiplication and permutation may each be done using a few APL characters.
4. APL handles collections of data as easily as individual items. For example, the volumes of five different cylinders may be calculated simultaneously by the same function used to calculate the volume of one.
5. Numbers used in APL are normally in familiar decimal form. Powers of 10 can be written using exponential or scientific notation, i.e., 1400000 can be written 1.4E6 or .14E7. Therefore, representation of very large or very small numbers is facilitated.

However, operators are provided to facilitate use of number systems other than base 10. This facility conforms to the modern mathematics curriculum used in many schools.

6. APL avoids the use of complex function hierarchy rules. The order of execution of an expression is determined first by the placement of parentheses and otherwise by one simple rule -- each operator works on everything to its right. For example, in the expression \( \frac{17 \div 2 + 3}{1} \), the \( \div \) has as its denominator \( 2 + 3 \). The entire expression may be read as \( 17 \div \) divided by the sum \( 2 + 3 \).

7. Correction of errors discovered before a statement is entered into the computer is simple. The system provides "visual fidelity" acting upon exactly what appears on the printed page, regardless of the order of entry. The printed record therefore makes clear exactly what was acted upon by the computer. This is important for later review and explanation by a teacher.

8. Error messages are concise and indicate to the student not only the type of error he has committed, but also where the error was made. Resumption of work from the point of error is simple.

9. APL\(\text{1130}\) provides a simple means for function definition. Special purpose operations needed for a particular problem may be easily defined by the student or by the teacher for use by the entire class.

10. APL\(\text{1130}\) provides an extensive library capability; each user can maintain a personal library where he may store and later retrieve programs, and also has the opportunity to use functions defined in other libraries.

11. APL\(\text{1130}\) incorporates card control commands which provide for punching APL functions and data on cards and for reading APL commands and statements from cards so punched, or from cards prepared on a separate keypunch.

OPERATING INSTRUCTIONS

This manual contains information required by those who manage and maintain an APL\(\text{1130}\) system. It is not needed by those who simply intend to use APL\(\text{1130}\). Part 1 of the User's Manual should also be consulted.

GENERATING THE SYSTEM

APL\(\text{1130}\) is written in 1130 Assembler Language and runs on a dedicated disk. It is independent of any other 1130 programming systems. To generate an APL\(\text{1130}\) system on a disk pack:

1. Initialize the disk, using either the Disk Pack Initialization Routine (DPIR) distributed with Version 1 of the Disk Monitor System or the Disk Cartridge Initialization Program (DCIP) distributed with Version 2.

2. Select the appropriate card decks for loading from the 1442 or 2501.

3. Place the IPL Sector deck in the card reader. Press START on the card reader and IMMEDIATE STOP, RESET and PROGRAM LOAD at the console. After the last card has left the hopper, press START on the card reader and PROGRAM START at the console so that the last card will be processed. Do not follow the deck with blank cards. Perform the last card procedure exactly as described.

4. Repeat step 3 with the APL System deck.

5. Repeat step 3 with the Empty Directories deck.

The APL system should now be ready for the first Initial Program Load, as described in the next section.

The APL System deck can be reloaded by itself if the system is destroyed or when a new release of APL\(\text{1130}\) is made. To do so, just repeat step 4. User enrollments and saved workspaces will be intact.
INITIAL PROGRAM LOAD

When the 1130 system is first turned on or when an APL disk has just been placed in the disk drive, an Initial Program Load procedure is needed to make the APL system operative. Two Initial Program Load cards are provided for this: one labeled APL1PL (for general use) and one labeled APL1PLPR (for "privileged" use - this card allows the use of the system maintenance commands by the first person who signs on).

The Initial Program Load Procedure is given in Part 1 of the User's Manual.

When an APL disk has been freshly generated, the first Initial Program Load should be performed with the privileged load card and the operator should sign on using the number 0. This allows him to assign the first users to the system. The number 0 need not be used after the first sign-on; additional users can be assigned at any time by a privileged user.

USE OF CONSOLE SWITCHES

Console switches 0 and 1 select the input device from which the next sign-on will be accepted. They may be set at any time and take effect at the next sign-on attempt, as follows:

Switch 0 down, switch 1 down: The 1131 is the APL terminal device. Typed input will be accepted from the console keyboard.

Switch 0 down, switch 1 up: The 1131 is the APL terminal device. Card input will be accepted from the card reader, as if a CARD command had been obeyed previously.

Switch 0 up: The 2741 is the APL terminal device. Typed input will be accepted from the 2741. Switch 1 is ignored, and card input will be accepted only upon execution of a CARD command.

SYSTEM MAINTENANCE COMMANDS

The system maintenance commands are intended for the use of those responsible for maintaining the APL 1130 system; they are not for general use. For that reason, they are available only to a privileged user, i.e., the first user to sign on after "privileged" IPL. They allow for the assigning of workspaces to users, the removal of users who will no longer use the system, and the printing of information about the current state of the APL system.

)ASSIGN N USERNAME [LOCK]

Assign user N (N is between 1 and 65535) another workspace (which may be his first) and call him USERNAME. The lock (a colon followed by a password; see discussion of locks and keys in the User's Manual) is optional. For example, )ASSIGN 3141 JSMITH or )ASSIGN 3141 JSMITH:LC1

Trouble reports:
SYSTEM FULL - All disk space has been assigned.
INCORRECT COMMAND

)EXFUMGE N

Remove user number N from the system, along with all of his workspaces, freeing the space to be assigned to other users.

Trouble reports:
NUMBER NOT IN SYSTEM - An attempt has been made to expunge a non-existent account number.
INCORRECT COMMAND

)PEOPLE

List all the users in the system, giving for each user his username, his account number, and the number of times he has signed on.

Trouble reports:
INCORRECT COMMAND
SPACES

List all the workspaces in the system, classified under
the users to whom they belong.

Trouble reports:
INCORRECT COMMAND

DROP WSID

This command has the same effect as the non-privileged
DROP command described in the User's Manual, except that
the WSID may contain any account number, and that using it
during a privileged sign-on decreases by one the number of
workspaces allotted to the account number (unless it has
only one workspace).

Sample Typewriter Output

>0
SIGNED ON APL/1130
>ASSIGN 100 TEACHER
>ASSIGN 100 TEACHER
>ASSIGN 1 STUDENTA
>ASSIGN 2 STUDENTS
SIGNED OFF

>100
TEACHER SIGNED ON

>3
SAVE WORK
WORK SAVED
OFF
SIGNED OFF

>2
NUMBER NOT IN SYSTEM

>1
STUDENTA SIGNED ON

>3
SAVE SOMEWHERE
SOMEWHERE SAVED
CLEAR

>2
SAVE THIS
WS RATION EXCEEDED

>1
SOMEBEHIND

>3
DROP SOMEWHERE
SOMEWHERE DROPPED

>1
SAVE THIS
THIS SAVED

>1
THIS
OFF
SIGNED OFF

Only when new system generated
is user number 0 used
User no. 100 named Teacher assigned
to system with two workspaces.
Users nos. 1 and 2 named Student
A & B assigned to system with one
workspace each.
After every sign off system in
student mode.
User no. 100 named Teacher signs on.
Performs some work

Saves workspace
User named Teacher signs off.
User no. 3 tries to sign on, but
has not been assigned to system.
User no. 1 signs on
Performs work

Saves workspace Somewhere.
Loads clean workspace
Performs more work
Tries to save This, but has already
used the one space he was assigned.
Lists his workspaces.
Drops previously saved Somewhere.
Lists again - there are no more
saved workspaces. Now can save This.

Lists saved workspaces. This is saved.
Signs off. System now waiting
for another student to sign on.
APL SYSTEM LOAD

Notes:
1. SEG 1-SEG 4 are loaded separately (see operating Instructions).

2. SEG 1 is only required to clear the disk. It may be omitted in rebuilding a pack.

3. SEG 4 (APL BLANK DIRECTORIES) may be omitted in rebuilding a pack where:
   a. former workspaces and/or functions are to be preserved.
   b. former directories are known to be unchanged.
ACKNOWLEDGEMENTS

The present APL\1130 Manual is based on the APL\360 Manual, just as the APL\1130 system is based on the APL\360 system. For assistance in preparing the necessary modifications, the authors are indebted to Messrs. S. M. Raucher*, D. J. Hills†, C. A. Weidman†, D. Oldacre‡, and L. M. Breed.

The original APL\1130 system was implemented by Messrs. R. S. Carberry, P. S. Abrams‡‡, L. M. Breed, C. H. Brenner, and A. G. Nemeth. Specifications for the present revision were drawn up by Messrs. Raucher, Breed, and Oldacre; the revisions were implemented by Mr. D. Oldacre and Mr. E. B. Ivenson§, with the assistance of Messrs. R. Kerley and L. M. Breed.

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.

* GEM Region, IBM Corporation, Washington, D. C.
† Amherst College, Amherst, Massachusetts
‡ I. P. Sharp Associates, Toronto, Canada
** Hartford GEM Office, IBM Corporation
†† Stanford University, Stanford, California
§§ IBM Corporation, White Plains, New York

TABLE OF CONTENTS

PART 1: GAINING ACCESS

THE APL CHARACTER SET
STARTING THE SYSTEM
STARTING AND ENDING A WORK SESSION
MISTAKES
CONNECTING A 2741 TERMINAL
Set up terminal, Dial computer, Transmission errors
LIMITED USE OF THE SYSTEM

PART 2: SYSTEM COMMANDS

WORKSPACES AND LIBRARIES
Workspaces, Libraries
NAMES
Local and global significance
LOCKS AND KEYS
ATTENTION
USE OF SYSTEM COMMANDS
Classification of commands, Normal responses and trouble reports, Summary
TERMINAL CONTROL COMMANDS
WORKSPACE CONTROL COMMANDS
Application packages, Information transfer between workspaces, Detailed description
LIBRARY CONTROL COMMANDS
Continuity of work, Workspace identification, Library and account numbers, Storage allotment, Purging a workspace, Detailed description
INQUIRY COMMANDS
Detailed description
CARD CONTROL COMMANDS
Preparing APL statements on cards, Detailed description
<table>
<thead>
<tr>
<th>PART 3: THE LANGUAGE</th>
<th>52</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FUNDAMENTALS</strong></td>
<td>52</td>
</tr>
<tr>
<td>Statements, Scalar and vector constants, Names and spaces, Overstriking and erasure, End of statement, Order of execution, Error reports, Names of primitive functions</td>
<td></td>
</tr>
<tr>
<td><strong>SCALAR FUNCTIONS</strong></td>
<td>56</td>
</tr>
<tr>
<td>Precision of numbers, Monadic and dyadic functions, Vectors, Index generator</td>
<td></td>
</tr>
<tr>
<td><strong>DEFINED FUNCTIONS</strong></td>
<td>60</td>
</tr>
<tr>
<td>Introduction, Branching, Local and global variables, Explicit argument, Explicit result, The forms of defined functions, Use of defined functions, Recursive function definition, Trace control</td>
<td></td>
</tr>
<tr>
<td><strong>MECHANICS OF FUNCTION DEFINITION</strong></td>
<td>67</td>
</tr>
<tr>
<td>Labels, Revision, Display, Line editing, Reopening function definition, Locked functions, Deletion of functions and variables, System command entered during function definition</td>
<td></td>
</tr>
<tr>
<td><strong>SUSPENDED FUNCTION EXECUTION</strong></td>
<td>70</td>
</tr>
<tr>
<td>Suspension, State indicator</td>
<td></td>
</tr>
<tr>
<td><strong>HOMONYMS</strong></td>
<td>72</td>
</tr>
<tr>
<td>Variable names, Function names</td>
<td></td>
</tr>
<tr>
<td><strong>INPUT AND OUTPUT</strong></td>
<td>73</td>
</tr>
<tr>
<td>Evaluation input, Character input, Normal output, Heterogeneous output</td>
<td></td>
</tr>
<tr>
<td><strong>RECTANGULAR ARRAYS</strong></td>
<td>76</td>
</tr>
<tr>
<td>Introduction, Vectors dimension catenation, Matrices dimension ravel, Reshape, Uses of empty arrays, Indexing, Indexing on the left, Array output</td>
<td></td>
</tr>
<tr>
<td><strong>FUNCTIONS ON ARRAYS</strong></td>
<td>82</td>
</tr>
<tr>
<td>Scalar functions, Reduction, Inner product, Outer product</td>
<td></td>
</tr>
<tr>
<td><strong>MIXED FUNCTIONS</strong></td>
<td>85</td>
</tr>
<tr>
<td>Introduction, Monadic transpose, Rotate, Reverse, Compress, Expand, Decode, Encode, Index of Membership, Take and drop, Grade up and down, Deal, Comments</td>
<td></td>
</tr>
<tr>
<td><strong>MULTIPLE SPECIFICATION</strong></td>
<td>91</td>
</tr>
</tbody>
</table>
LIST OF ILLUSTRATIONS

Table 1.3 TELEPHONE NUMBERS
Table 2.1 SYSTEM COMMANDS
Table 2.2 MNEMONICS & CARD CODES FOR THE APL CHARACTER SET
Table 3.1 ERROR REPORTS
Table 3.2 PRIMITIVE SCALAR FUNCTIONS
Table 3.3 FORMS OF DEFINED FUNCTIONS
Table 3.4 DIMENSION AND RANK VECTORS
Table 3.5 IDENTITY ELEMENTS OF PRIMITIVE SCALAR DYADIC FUNCTIONS
Table 3.6 INNER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTIONS f and g
Table 3.7 OUTER PRODUCTS FOR PRIMITIVE SCALAR DYADIC FUNCTION g
Table 3.8 PRIMITIVE MIXED FUNCTIONS

PART 1

GAINING ACCESS

The APL/I130 system employs the APL language described in Part 3 and the system commands described in Part 2. It is operated either from the I131 console or from an auxiliary 2741 terminal. This part of the manual describes the procedures for beginning and ending a work session.

The following physical equipment is required for the operation of APL/I130:

- An I131 CPU with 8K or more of core storage and a built-in disk drive.
- A 2501 card reader or 1442 card read/punch wired for Initial Program Load.
- An APL typing element I1167988.

Recommended, but not required, is a 2741 terminal and an IBM 1130 Communications Channel Adapter (RPQ W16427) with data-phone or IBM 4-wire modem, and 2741 interrupt feature. With the 2501 reader a 1442 punch is required for the }PCB and }PCBS commands but is not otherwise needed.

THE APL CHARACTER SET

The APL character set is shown in two arrangements in Figure 1.1; the first is appropriate to the 2741 terminal, and the second to the I131 console. On the 2741 terminal, 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.
Figure 1.1
This may be appearance (ω over W), Greek-Roman equivalence (ο over R), sequence (α ≤ ω > * over 3 ≤ 5 7 8), or some -- possibly farfetched -- relationship between the APL function represented by the symbol and the letter (α over P for power, ' over X for "wote", and I over S for ceiling).

Because of the smaller number of keys available, the 1131 console uses a three-case system, also shown in Figure 1.1. The currently active case is indicated by the left and right halves of the ACCUMULATOR and ACCUMULATOR EXTENSION lights in the manner shown in Figure 1.2.

The two keys at the upper left control the cases, the arrows on the keys indicating the transitions (from the active case to the new active case) initiated by striking them. If the shift is in either upper or middle case but not locked, it may be locked in that state by a further depression of the shift key which brought it to that state. If the shift is locked in upper or middle case, it remains in that state until returned to lower case by the action of the appropriate shift key or by a carrier return. The 1131 Alpha/Numeric shift must remain in Alpha shift at all times.

For the console printer, the APL character set is provided by printing element #1167988. A particular 2741 terminal may use either the #1167988 element (for terminals using PTTC/BCD printing elements) or the #1167987 element (for terminals using Selectric printing elements).

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.
STARTING THE SYSTEM

To start the APL system, turn the 1130 on, load the APL disk, mount the APL typing element (#1167988), and perform an Initial Program Load as follows:

1. Set console entry switch 0 down if the first sign-on is to come from the console keyboard, up if it is to come from the 2741 keyboard. Set all other console switches down.
2. Place the APLFL card (APL initial program load) in the hopper.
3. Press IMMEDIATE STOP, RESET, and PROGRAM LOAD at the console.

The system will then be ready for the first sign-on.

STARTING AND ENDING A WORK SESSION

This section describes the procedures for beginning and ending a work session; more detail is given in Part 2.

The following procedures apply to operations from either the console keyboard or from a 2741 terminal. In the case of the 2741, a connection must first be established in the manner described in the succeeding section.

Communication with the computer is carried on by means of entries from the console keyboard (or auxiliary 2741 terminal), 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 and strike the carrier return to indicate the end of the message.

Each user is assigned an account number. This number is used in the sign-on that begins a work session and serves to partially identify any work that the user may store in the system.

To begin a session, enter a right parenthesis followed by the appropriate account number, followed by a carrier return. For example:

)581

The account number may include a key (i.e., a colon and a password). For example:

)581:ABC

If the foregoing is properly executed and if the account number has previously been entered into the system (as described in the operator's manual), the computer will respond by typing the name associated with the account number, followed by SIGNED ON and the name of the system, e.g.:

SMITH SIGNED ON

APL \ 1130

If the sign-on is incorrect, one of the following trouble reports will be given:

INCORRECT SIGN-ON means that the form of the command was faulty.

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.

INCORRECT COMMAND means that some account number is already signed on. Sign off, then execute the foregoing again.

To end a terminal session, enter the command:

)OFF

The computer will respond by typing:

SIGNED OFF

In the remainder of this manual the need for carrier return will not be explicitly mentioned, since it is required for every entry.

Mistakes. Before the carrier return 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 and INT REQ on the 1131 console). 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.

If the keyboard of a 2741 terminal (either direct-connected or telephone-connected) simply unlocks after the carrier return in an attempted sign-on, repeat the sign-on procedure.
CONNECTING A 2741 TERMINAL

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

EC1. Set up terminal:
Insert paper, mount an APL printing element, connect terminal to power source, and set switches as follows:

<table>
<thead>
<tr>
<th>LCL/COM</th>
<th>COM</th>
<th>Power</th>
<th>ON</th>
</tr>
</thead>
</table>

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.

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.

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.

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 by typing RESEND. The last entry from the keyboard should then be repeated.

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.

LIMITED USE OF THE SYSTEM

No system commands other than the sign-on and sign-off described here are required in order to make use of Part 1, and the reading of Part 2 may therefore be deferred if only casual or restricted use is to be made of the system.

Table 1.3: TELEPHONE NUMBERS

<table>
<thead>
<tr>
<th>123 456-7890</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert the access telephone number here.</td>
</tr>
<tr>
<td>An assistance number should be included.</td>
</tr>
</tbody>
</table>

APL Operator: 123 456-7899
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\1130 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\1130 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\1130 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 present when the system is signed on. All transactions with the system are mediated by it. 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, and information on the progress of program execution is maintained in the state indicator of 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 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. However, one user may activate a copy of another user's (unlocked) workspace if he knows the library number and workspace name.

Names

Names of workspaces, functions and variables may be formed of any sequence of alphabetic (A to Z) and numeric (0 to 9) characters that starts with an alphabetic and contains no blank. However, workspace names may not exceed 11 characters in length, and other names may not exceed six.

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., 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, 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锁, 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 never give the keys, and do not overtly indicate the existence of a lock.

Account numbers can be similarly protected by locks and keys, to avoid their unauthorized use and maintain the integrity of a user's private library.

Passwords for locks and keys may be formed of any sequence of alphabetic and numeric characters up to six characters long, without blanks. In use, 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.

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 INT REQ on the 1131 console keyboard. (The same key is used for linefeed when the keyboard is not locked.)

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). In some cases a line will be printed before the keyboard unlocks, telling where a function in progress was interrupted.

Except for card I control commands, the execution of system commands, once entered, cannot be interrupted.

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.

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. Card control commands allow the reading of APL statements from cards rather than the keyboard, and the punching of variables and functions into cards for later use.
<table>
<thead>
<tr>
<th>Reference and Purpose</th>
<th>NORMAL RESPONSE</th>
<th>TROUBLE REPORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>USER SIGNED ON; SYSTEM</td>
<td>1 8</td>
</tr>
<tr>
<td>TC2</td>
<td>SIGNED OFF</td>
<td>11</td>
</tr>
<tr>
<td>WC1</td>
<td>CLEAR</td>
<td>11</td>
</tr>
<tr>
<td>WC2</td>
<td>WSID LOADED</td>
<td>9 11</td>
</tr>
<tr>
<td>WC3</td>
<td>NAMES</td>
<td>2 4 7 9 10 11</td>
</tr>
<tr>
<td>WC3a</td>
<td>NAMES</td>
<td>2 7 9 10 11</td>
</tr>
<tr>
<td>WC4</td>
<td>NAMES</td>
<td>2 4 7 9 10 11</td>
</tr>
<tr>
<td>WC4a</td>
<td>NAMES</td>
<td>2 7 9 10 11</td>
</tr>
<tr>
<td>WC5</td>
<td>ERASE NAME[S]</td>
<td>3 11</td>
</tr>
<tr>
<td>LC1</td>
<td>SAVE WSID [LOCK] WSID SAVED</td>
<td>5 6 7 11</td>
</tr>
<tr>
<td>LC2</td>
<td>DROPPED WSID</td>
<td>9 11</td>
</tr>
<tr>
<td>IQ1</td>
<td>FUNCTION NAMES</td>
<td>11</td>
</tr>
<tr>
<td>IQ2</td>
<td>VARIABLE NAMES</td>
<td>11</td>
</tr>
<tr>
<td>IQ3</td>
<td>SEQUENCE OF HALTED FUNCTIONS</td>
<td>11</td>
</tr>
<tr>
<td>IQ4</td>
<td>SEQUENCE OF HALTED FUNCTIONS WITH LOCAL NAMES</td>
<td>11</td>
</tr>
<tr>
<td>IQ5</td>
<td>WSID</td>
<td>11</td>
</tr>
<tr>
<td>IQ6</td>
<td>NAMES OF STORED WORKSPACES</td>
<td>8 11</td>
</tr>
<tr>
<td>CD1</td>
<td>CARD [NUMBER]</td>
<td>11</td>
</tr>
<tr>
<td>CD1a</td>
<td>CARD [NOEDIT] [NO Disp]</td>
<td>11</td>
</tr>
<tr>
<td>CD2</td>
<td>CARD END</td>
<td>11</td>
</tr>
<tr>
<td>CD2a</td>
<td>CARD END</td>
<td>11</td>
</tr>
<tr>
<td>CD3</td>
<td>PCH NAMES</td>
<td>4 7 11</td>
</tr>
<tr>
<td>CD3a</td>
<td>PCH</td>
<td>7 11</td>
</tr>
<tr>
<td>CD4</td>
<td>PCHS WSID [KEY] NAMES</td>
<td>4 7 9 11</td>
</tr>
<tr>
<td>CD4a</td>
<td>PCHS WSID [KEY]</td>
<td>7 9 11</td>
</tr>
</tbody>
</table>

Notes: 1. Items in brackets are optional.
2. KEY or LOCK: a password preceded by a colon.
3. WSID: library number and workspace name, or workspace name alone.
4. CD1 and CD2a are from the keyboard; CD1a and CD2 are from cards.
5. A stop is an attention signal, or a card with )CARD END or )OFF.
6. Trouble report forms:
   1 INCORRECT SIGN ON 7 NOT WITH OPEN DEFN
   2 NOT COPIED NAMES 8 NUMBER NOT IN SYSTEM
   3 NOT ERASED NAMES 9 WS NOT FOUND
   4 NOT FOUND NAMES 10 WS FULL
   5 NOT SAVED WS IS WSID 11 INCORRECT COMMAND
   6 NOT SAVED WS QUOTA USED UP

Table 2.1: SYSTEM COMMANDS
The text that follows is based upon this classification, although it will be seen that 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 completely 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.

Summary. The purposes, forms, responses, and trouble reports for all system commands are summarized in Table 2.1.

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.

TERMINAL CONTROL COMMANDS

There is one command for starting, and one command for ending, a work session. The starting command has been described in Part 1.

ACTION

TC1. Start a work session: Enter SIGN-ON, described in Part 1.

NOTES

See Part 1, Starting and Ending A Work Session.

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

Effect:

1. The currently active workspace will vanish.

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

Response:

1. SIGNED OFF

Trouble report:

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; and remove unwanted objects from the active workspace. 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\1130 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.

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 selection of objects (i.e., functions or variables) from it. When an entire workspace is copied, all the functions and global variables within it are subject to the operation, but its state indicator and local variables are left behind.

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.

ACTION

WC1. Activate a clear workspace:
Enter CLEAR.

Effect:
1. A clear workspace will be activated, replacing the presently active workspace.

Response: None.

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: The workspace name, followed by LOADED, is printed.

NOTES

This command is used to make a fresh start, discarding whatever is in the active workspace.

A clear workspace has no variables or defined functions.

Its workspace identification does not match that of any stored workspace. (See section on library control).

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 either that there is no stored workspace with the given identification, or that no key, or the wrong key, was used when one was required.

INCORRECT COMMAND
WC3. Copy global objects from a stored workspace:
Enter `COPY`
followed by a space and a
workspace identification
(with the key, if required),
and the names of objects to
be copied, separated by
spaces.

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

Response: None.
Response:
1. NOT COPIED, followed by the names of objects not copied, will be printed if appropriate.

Trouble reports:
NOT WITH OPEN DEFN
NOT FOUND
WS FULL
See WC3 for meanings.

WS NOT FOUND
See WC2 for meaning.
INCORRECT COMMAND

WC4a. Copy all global objects from a stored workspace, protecting the active workspace:
Enter )COPY
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. NOT COPIED, followed by the names of objects not copied, will be printed if appropriate.

See note at WC3, Effect 1.

WC5: 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: None.

Trouble report:
NOT ERASED
followed by the names of functions not erased, means those functions are pendent.
INCORRECT COMMAND

LIBRARY CONTROL COMMANDS

There are two basic operations performed by the commands in this class. The save command causes 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 command 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. A clear workspace activated by a clear command or a sign-on, is called CLEAR WC, which cannot be the name of a stored workspace.

The identification of active workspaces is used 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, will be stopped.

Library and account numbers. A user's account number is also the number of his private library.

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 contents of another user's private library. 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/1130 is assigned library space in terms of the maximum number of stored workspaces he may have at one time. The allotment for each user is determined by those responsible for the general management of a particular system, within the bounds of the physical resources of the system.

Purging a workspace. The sequence of commands, SAVE ABC123, CLEAR, COPY ABC123, will purge the active workspace, clearing it of all but its functions and global variables. This often results in more usable space than can otherwise be realized.

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

ACTION

LCL. 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.

Response:
1. The workspace name, followed by SAVED, will be printed.

NOTES

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.

Trouble reports:
NOT WITH OPEN OPEN 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 first dropping a workspace already stored.

NOT SAVED 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.

INCORRECT COMMAND
LG2. Erase a stored workspace.
Effect: The designated stored workspace will be expunged. 
Response: The workspace name, followed by DROPPED, will be printed.

Trouble reports:
1. The workspace name, followed by DROPPED, will be printed.

INCOMPLETE COMMAND

IQ2. List names of global variables:
Enter JVARS
Effect: None.
Response:
1. The names of global variables in the active workspace will be printed.

Trouble report: INCOMPLETE COMMAND

IQ3. List halted functions:
Enter JSI
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. Suspended functions will be distinguished from pending functions by an asterisk.

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

Trouble report: INCOMPLETE COMMAND

IQ4. List halted functions with names of local variables:
Enter JSIV
Effect: None.
Response:
1. The response will be the same as for IQ3, except that with each function listed there will appear a listing of its local variables.

Trouble report: INCOMPLETE COMMAND

IQ5. Give identification of active workspace:
Enter JACID
Effect: None.
Response:
1. The identification of the active workspace will be printed.

Trouble report: INCOMPLETE COMMAND
A library number is not required for listings of the user's private library.

Effect: None.

Trouble report
NUMBER NOT IN SYSTEM

INCORRECT COMMAND

CARD CONTROL COMMANDS

The commands in this class provide for the use of punched cards. They allow the user to:

1. Prepare APL statements and system commands on an 029 keypunch for future entry to APL/1130, using the \CARD and \CARD END commands.

2. Have the APL system punch copies of variables and functions into cards for later use, using the \PR CH and \PR FH commands.

Preparing APL statements on cards. Statements for input to the APL/1130 system may be punched into cards in free format in columns 1-71. If a statement is too long to fit onto one card, any non-blank character can be punched in column 72 to indicate that the statement continues on the next card. Columns 73-80 are ignored when the cards are being read, and can be used for identification or sequence numbers. They are so used by the punch commands, which place the name of the variable or function in columns 73-78, and the sequence numbers for that name in columns 79 and 80.

A list of APL characters and their keypunch equivalents is given in Table 2.2. All the letters and digits and many of the APL operators (e.g., +, /, ?, =) are available as single characters on the keypunch. The symbols -, [ ], ;, " and - are represented on the keypunch by substitute characters, e.g., - is represented by $.
For other non-alphanumeric characters a system of mnemonic names is used. These mnemonics always consist of an at-sign (@) followed by a string of letters designed to suggest the name or use of the symbol. Thus the mnemonic for $ is @RHO, the mnemonic for $ can be either @MAX or @CEIL, and so on. Mnemonics may be used in APL statements in exactly the way the corresponding symbols would be used, except that a blank must be left at the end of the mnemonic. For instance, the statement $Q would be punched in a card as $@RHO Q.

Detailed description. The term 'stop' is used here to mean the occurrence of a card punched with )CARD END or )OFF, or an attention signal, while cards are being read.

**ACTION**

CD1. Initiate reading of cards.
Enter )CARD followed, if desired, by one of each of the pairs: EDIT or NEDIT, DISP or NODISP.

**NOTES**

This command enables the system to accept APL statements and system commands from the card reader, rather than from the keyboard. All statements and commands except )PCS and )PCES may be executed from both sources.

**Effect:**

1. The keyboard will lock.
2. The system will accept instructions from the card reader until a stop occurs or an error is encountered.

If EDIT has been specified, the keyboard unlocks immediately when an error occurs or an attention signal is given. If NEDIT is specified, the cards are flushed up to a stop before the keyboard is unlocked. (This is the same as the effect of CD1.)

CD2a. Continue card reading with specified edit and display options.
Include in the card deck a card punched with )CARD followed, if desired, by one of each of the pairs: EDIT or NEDIT, DISP or NODISP.

**Effect:**

1. The edit or display options will be set appropriately.

**Response:** None.

Trouble report
INCORRECT COMMAND

**CD2. Stop reading cards and return control to the keyboard:**
Include in the card deck a card punched with )CARD END.

**Effect:**

2. The keyboard unlocks.

**Response:** None.

Trouble report
INCORRECT COMMAND

**CD2a. Run cards through to stop and return control to the keyboard:**
Enter )CARD END

**Effect:**

1. Cards are read but instructions are not executed until a stop occurs.
2. The keyboard unlocks.

**Response:** None.

Trouble report
INCORRECT COMMAND
CD3. Punch cards for global objects in the active workspace:
Enter )PCB
followed by a space and one or more names of objects, separated by spaces.

Effect:
1. A copy of each named object is punched into cards.

Locked functions (see Part 3) cannot be punched.

Response: None.

Trouble reports
NOT WITH OPEN DEFN
NOT FOUND
See note at CD3.

CD3a. Punch cards for all global objects in the active workspace:
Enter )PCB

Effect:
1. A copy of each function and global variable in the active workspace is punched into cards.

Response: None.

Trouble reports
NOT WITH OPEN DEFN
See CD3 for meaning.

INCORRECT COMMAND

CD4. Punch cards for global objects in a stored workspace.
Enter )PCBS
followed by a space and the workspace identification (with the key, if required), and one or more names of objects, separated by spaces.

Effect:
1. A copy of each named object is punched into cards.

Response: None.

Trouble reports
NOT WITH OPEN DEFN
NOT FOUND
See CD3 for meanings.

INCORRECT COMMAND

WS NOT FOUND
means either there is no stored workspace with the given identification, or the wrong key (or no key) was used when one was required.
CD4a. Punch cards for all global objects in a stored workspace:
Enter }PCNS followed by a space and the workspace identification (with the key, if required).

**Effect:**
1. A copy of each function and global variable in the designated workspace is punched into cards.

**Response:** None.

**Trouble reports**
- NOT WITH OPEN DEFN
  See CD3 for meaning.
- WS NOT FOUND
  See CD4 for meaning.
- INCORRECT COMMAND

---

**PART 3**

**THE LANGUAGE**

The APL/1130 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 ÷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.

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 + and treated in the section on Defined Functions), and the *specification*. A typical specification statement is of the form

\[ x = 3 \times 4 \]

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:

\[ \text{3} \times \text{4} \]

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 \( \times \) followed immediately by an integer. The integer following the \( \times \) specifies the power of ten by which the part preceding the \( \times \) is to be multiplied. Thus \( 1.44 \times 2 \) is equivalent to \( 1.44 \).

Negative numbers are represented by a negative sign immediately preceding the number, e.g., \( -1.44 \) and \( -1.44 \times 2 \) 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 "CANNOT" 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 six or fewer letters or digits beginning with a letter and not containing a space.

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 \( 3 \times 4 \) may be entered with no spaces, but if \( x \) is a defined function, then the expression \( 3 \times x \) 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. \( + \) and \( \times \) ), and
3. to position the carriage for erasure, which is effected by striking the linefeed (marked ATTN on IBM 2741 terminals and INT REQ on the 1131 console keyboard). 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. 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 \( 3 \times 4 + 6 \times 2 \), 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 \( w = (3 \times (x+y)-z) \), 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 \( w = (3 \times (x+y)-z) \).

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.
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 followed by a zero (-0), 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 3.6, 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 \( \rho M \) yields the dimension of the array \( N \), but the terms \( \text{size} \) or \( \text{shape} \) may be preferred both for their brevity and for the fact that they avoid potential confusion with the \text{dimensionality} or \text{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., "\( \rho \)" for "\text{size}", and "\( \iota \)" for "\text{index generator}").

SCALAR FUNCTIONS

Each of the primitive functions is classified as either \text{scalar} or \text{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.

Precision of numbers. Integers less than 2 to the power 23 are carried with full precision; larger numbers and non-integers are carried to a precision of 6 to 7 decimal digits.
<table>
<thead>
<tr>
<th>Monadic form fB</th>
<th>Dyadic form AfB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition or example</strong></td>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>+B ↔ 0+B</td>
<td>Plus</td>
</tr>
<tr>
<td>-B ↔ 0-B</td>
<td>Negative</td>
</tr>
<tr>
<td>×B ↔ (B&gt;0)-(B&lt;0)</td>
<td>Signum</td>
</tr>
<tr>
<td>÷B ↔ 1+B</td>
<td>Reciprocal</td>
</tr>
<tr>
<td>B</td>
<td>Ceiling</td>
</tr>
<tr>
<td>-3.14</td>
<td>Floor</td>
</tr>
<tr>
<td><strong>Exponential</strong></td>
<td></td>
</tr>
<tr>
<td>•N ↔ N ↔ •N</td>
<td>Natural logarithm</td>
</tr>
<tr>
<td></td>
<td>-3.14 ↔ 3.14</td>
</tr>
<tr>
<td>!0 ↔ 1</td>
<td>Factorial</td>
</tr>
<tr>
<td>!B ↔ B×!B-1</td>
<td>Roll</td>
</tr>
<tr>
<td>?B ↔ Random choice from \B</td>
<td>Pi times†</td>
</tr>
<tr>
<td>OB ↔ B×3.14159...</td>
<td>Not</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(-A)OB</th>
<th>A</th>
<th>AOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1-B×2)*.5</td>
<td>0</td>
<td>(1-B×2)*.5</td>
</tr>
<tr>
<td>Arccos 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>Arccosh 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>

Table of Dyadic o Functions

†This function not available on APL\1130

Table 3.2: PRIMITIVE SCALAR FUNCTIONS

57
For functions such as floor and ceiling, and in comparisons, a "fuzz" of about $10^{-7}$ 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 $=\neq$, 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, $\lfloor x \rfloor$ denotes the monadic function ceiling applied to the single argument $x$, and $\lfloor x \rfloor y$ 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 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 $A \mod B$ 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 $R = N \times A$, 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, $1 \mod 2.6$ is 0.6 and $1 \mod 0.6$ is 0.6.

In APL,$\backslash 130\$, the domain of the $\cong$ function is limited to positive integer arguments less than 32768.

The factorial function $!B$ is defined in the usual way as the product of the first $N$ positive integers. The function $A \mod B$ (pronounced $A$ out of $B$) is defined as $(\lfloor A/B \rfloor + 1) \times B - A$ and is the number of combinations of $B$ things taken $A$ at a time.

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

\[ \begin{array}{c}
3 \equiv 7 \\
1 \\
7 \equiv 3 \\
0
\end{array} \]

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, $\equiv$ is equivalent to material implication, and $\equiv$ is equivalent to exclusive-or. These six functions together with the and, or, and, 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 vector (or a single element vector or matrix) and a vector of any dimension, the scalar being used with each component of the vector. For example:

\[ \begin{array}{c}
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
7 \\
8 \\
9 \\
10
\end{array} \]

Index generator. If $N$ is a non-negative integer, then $1 \div N$ denotes a vector of the first $N$ integers. The dimension of the vector $1 \div N$ is therefore $N$; in particular, $1 \div 1$ is a vector of length one which has the value 1, and $1 \div 10$ is a vector of
length zero, also called an empty vector. The empty vector prints as a blank. For example:

\[
\begin{array}{c}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{array}
\]

Empty vector prints as a blank

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

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.

DEFINITE 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

\[
\begin{array}{l}
\text{VSPHERE} \\
[1] \text{SURF=4×3.14159×R×R} \\
[2] \text{VOL+SURF×R+3} \\
[3] \text{V}
\end{array}
\]

is called a function definition; the first \(V\) (pronounced \(VEL\)) marks the beginning of the definition and the second \(V\) marks the conclusion; the name following the first \(V\) (in this case \text{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. In APL.1130, the number of statements in a function body may not exceed 50.

The act of defining a function neither executes nor checks for validity the statements in the body. After the definition of a function is completed, entering the name of the function causes the execution of the statements in the body. For example:

\[
\begin{array}{l}
\text{VSPHERE} \\
[1] \text{SURF=4×3.14159×R×R} \\
[2] \text{VOL+SURF×R+3} \\
[3] \text{V}
\end{array}
\]

Definition of the function \text{SPHERE}

\[
\begin{array}{l}
\text{R=2} \\
\text{SURF} \\
\text{VALUE ERROR} \\
\text{SURF} \\
\text{SPHERE} \\
\text{SURF} \\
\text{50.2654} \\
\text{VOL} \\
\text{33.5103} \\
\text{R=1} \\
\text{SPHERE} \\
\text{SURF} \\
\text{12.5664} \\
\text{VOL} \\
\text{4.18879}
\end{array}
\]

Specification and display of the argument \(R\)

Execution of \text{SPHERE}

SURF and VOL now have values assigned by the execution of \text{SPHERE}

Use of \text{SPHERE} for a new value of the argument \(R\)

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 \(\text{V} \rightarrow \text{V}\) denotes a branch to statement \(V\) and causes statement \(V\) 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:

1. SUM
   2. S=0
   3. I=I+1
   4. +4×I≤N
   5. I=I+1
   6. +3
   7. N=1
   8. SUM
   9. S
   10. N=2
   11. SUM
   12. S
   13. N=5
   14. SUM
   15. Equivalent to SUM
   16. V
   17. S

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 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 SUM2, which has a local variable I, with the behavior of the function SUM3 in which I is global:

1. VSUM3;I
2. [1] S=0
3. [2] I=0
5. [4] I=I+1
6. [5] +3×I≤N
8. [7] I=20
9. [8] N=5
10. [9] SUM
11. [10] SUM2
13. [12] S
15. [14] I
17. [16] I

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.

62
63
**Explicit argument.** A function definition of the form

```
VSPH X
```

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
```

50.2054

```
SPH 1
```

12.5664

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 \( VZ \) produces an explicit result and may therefore appear in a compound expression such as \( X+VZ \). A function definition of the form

```
VZ=SPH X
```

defines \( SPH \) as a function with an explicit result; the variable \( E \) 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*SPH 1
```

37.6991

```
R=2
```

```
(SP R)*R+3
```

39.5103

**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 \( 2=F \) is invalid.

<table>
<thead>
<tr>
<th>Number of Arguments</th>
<th>Number of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VY</td>
</tr>
<tr>
<td>1</td>
<td>V3=F Y</td>
</tr>
<tr>
<td>2</td>
<td>VY F Y</td>
</tr>
</tbody>
</table>

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:

```
VZ=SQRT X
```

[1] \( \rightarrow 0 \times X < 0 \)

[2] \( 2-X+0.5 \)

Q=SQRT 16

Q

4

Q=SQRT -16

VALUE ERROR

Q=SQRT -16

A
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 SQT:

\[
\begin{align*}
VZ &= SQT X \\
Z &= X + 5V \\
VS &= A + B \\
VS &= SQT (A^2 + B^2) \\
5 &= HYP 12
\end{align*}
\]

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:

\[
\begin{align*}
VZ &= FAC \ N \\
Z &= X \times \text{B} \\
Z &= B + FAC \ N-1 \\
Z &= 0 \\
Z &= 1V
\end{align*}
\]

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 \( TAP \). If one types \( TAP = 2 \) \( 3 \) \( 5 \) 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 \( TAP = 0 \) will discontinue
tracing of \( P \). A complete trace of \( P \) is set up by entering
\( TAP = N \), where \( N \) is the number of statements in \( P \).

MECHANICS OF FUNCTION DEFINITION

When a function definition is opened (by typing \( a \) \( V \)
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 name 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. A label is a local name as if it had occurred in
the header; unlike a local variable it cannot be
respecified.

Revision. Any statement number (including one typed by the
system) can be overridden by typing \( [N] \), where \( N \) is any
positive number less than 100, with or without a decimal
point and with at most two digits to the right of the
decimal point. If \( N \) is zero, it refers to the header line
of the function.

If a statement number is used again, the new text
associated with it replaces the old 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 v 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 v 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 [M]. After the display, the system awaits replacement of statement N. Typing [D] displays the entire function, including the header and the opening and closing v, and awaits entry of the next statement; typing [MN] 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 [NUM] 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 without the line number. 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+R spaces, where R is the position 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.

A new statement number (in brackets) can be entered in the space left for it 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 vR alone; the rest of the function header must not be entered. The system responds by typing [[R+1]], where R 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, vR(3)X+1v initiates editing by entering a new line 3 immediately. The system responds by typing [+] and awaiting continuation. The entire process may be accomplished on a single line. Thus, vR(3)X+1v opens the definition of R, enters a new line 3, and terminates the definition mode. Also, vR([]P 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, vR([[)])v or vR([]) or vR[([)].

Locked functions. If the symbol v (formed by a v overstruck with a ~ and called del-tilde) is used instead of v 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 the type of error, not the statement. Finally, the trace control vector for a function cannot be changed after the function is locked.

Function locks 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). A variable also may be deleted 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 vCOPY, will be rejected with the message NOT WITH OPEN DEFN (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 two ways: by an error report, or by an attention signal. In either case, the function is still active and its execution can later be resumed. In this state the function is said to be suspended. Typing \(+K\) 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.

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 \( +N \)).
3. reopen the definition of any function which is not pendent. The term pendent is defined in the discussion of the state indicator below.

State indicator. Typing \( \text{SI} \) causes a display of the state indicator; a typical display has the following form:

\[
\text{SI}
\]

\[
H(7) \ *
\]

\[
G(2) \ *
\]

\[
F(3)
\]

The foregoing display indicates that execution was halted at 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 \( F \). (No line number is printed for locked functions). 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 \( G \), itself invoked in statement 8 of \( G \), a subsequent display of the state indicator would appear as follows:

\[
\text{SI}
\]

\[
G(5) \ *
\]

\[
G(8) \ *
\]

\[
H(7) \ *
\]

\[
G(2) \ *
\]

\[
F(3)
\]

It is recommended that the state indicator be cleared before modifying a program that uses statement labels. Changing the values of statement labels (by adding or removing statements) in the function will not affect the label values for a suspended execution, and if execution of the suspended function is continued, branch instructions may result in branches to the wrong statements.

The state indicator can often be cleared by repeated entry of \( \rightarrow 0 \). If this does not work, it can be cleared by a sequence of commands of the following form:

\[
\text{SAVE A } \text{CLEAR A COFF A}
\]

Trace control vectors may be set within functions. In particular, they may be set by expressions which initiate or discontinue tracing according to the 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 names. 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 (including labels). A sample display follows:

)SIV
P[4]  P J
Q[3]  C X T
R[3]  P
S[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 )PBC or )VARS.

As the state indicator is cleared 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 ++ 4  ||  ||  ||  ||
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 F 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 F inaccessible. However, even in such circumstances, the opening of function definition for such a function F 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:

\[
\begin{array}{ll}
\text{VZ+F X} & \text{VZ+F Y} \\
\text{Z-X+Y} & \text{Z-F Y} \\
\text{VZ+G Y} & \text{VZ+G R} \\
\text{Z-F Y} & \text{Z-F R} \\
\text{Y+3} & \text{Y+3} \\
\text{G 4} & \text{G 4} \\
\end{array}
\]

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:

\[
\text{VZ+A CPI B} \\
\text{Z+4*(1+0.01*B[1])*B[2]v}
\]

For example:

\[
\begin{array}{l}
1000 CPI 5 4 \\
1215.51
\end{array}
\]
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 1215.51

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.

Evaluation 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 to alert the user to the type of input expected, 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:

$$\begin{align*}
&2=\text{□}
\quad 2+q\times\text{□}\times 2 \\
&\text{□:} \quad 3 \\
&36 \\
&\text{□:} \quad 3\times 2
\end{align*}$$

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.

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:

$$\begin{align*}
&x=\text{□} \\
&\text{CAN'T} \\
&x \\
&\text{CAN'T}
\end{align*}$$

(Quote-quad input, not indented)

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\times x^2 assigns to q the value x^2 and then prints the value of x^2.
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=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:

VCI;A;I;Y
[1] 'ENTER CAPITAL AMOUNT IN DOLLARS'
[3] 'ENTER INTEREST IN PERCENT'
[4] I+1
[5] 'ENTER PERIOD IN YEARS'
[6] I+1
[7] 'RESULT IS ';AX(1+.01X1)*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. In APL/1130, arrays of rank greater than rank 2 cannot be used and no dimension of an array may exceed 256; thus, a vector may have no more than 255 elements and a matrix may have no more than 255 rows or columns.

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 pX denotes its dimension. For example, if X=2 3 5 7 11, then pX is 5, and if Y='ABC', then pY is 3. A single character entered in quotes or in response to a Y 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
Y=3 5 7 11

In general, the dimension of X,Y is equal to the total number of elements in X and Y.

Matrices, dimension, ravel. The monadic function p 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 pA is the vector 3 4.

Since pA contains one component for each coordinate of A, the expression pA is the rank of A. Table 3.4 illustrates the values of pA and pA for arrays of rank 0 (scalars) up to rank 2. In particular, the function p applied to a scalar yields an empty vector.

<table>
<thead>
<tr>
<th>Type of Array</th>
<th>pA</th>
<th>pA</th>
<th>pA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Vector</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Matrix</td>
<td>M</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.4: DIMENSION AND RANK VECTORS
The monadic function \texttt{ravel} is denoted by a comma; when applied to any \texttt{array} \texttt{A} it produces a \texttt{vector} whose elements are the elements of \texttt{A} in row order. For example, if \texttt{A} is the matrix
\begin{verbatim}
  2 4 6 8
10 12 14 16
18 20 22 24
\end{verbatim}
and if \texttt{V} is a \texttt{vector} of dimension 12 whose elements are the integers 2 4 6 8 10 12 ... 24. If \texttt{A} is a \texttt{vector}, then \texttt{.A} is equivalent to \texttt{A}; if \texttt{A} is a \texttt{scalar}, then \texttt{.A} is a \texttt{vector} of dimension 1.

\texttt{Reshape}. \ The dyadic function \texttt{ reshape} reshapes its right argument to the dimension specified by its left argument. If \texttt{M} is a \texttt{vector}, then \texttt{M} is an \texttt{array} of dimension \texttt{D} whose elements are the elements of \texttt{M}. For example, \texttt{2 3} \texttt{1 2} \texttt{3} \texttt{4} \texttt{5} \texttt{6} is the matrix
\begin{verbatim}
  1 2 3
  4 5 6
\end{verbatim}

If \texttt{N}, the total number of elements required in the \texttt{array} \texttt{DpV}, is equal to the dimension of the \texttt{vector} \texttt{V}, then the \texttt{ravel} of \texttt{DpV} is equal to \texttt{V}. If \texttt{N} is less than \texttt{.V}, then only the first \texttt{N} elements of \texttt{V} are used; if \texttt{N} is greater than \texttt{.V}, then the elements of \texttt{V} are repeated cyclically. For example, \texttt{2 3} \texttt{1 2} is the matrix
\begin{verbatim}
  1 2 1
  2 1 2
\end{verbatim}
and \texttt{3} \texttt{3} \texttt{1 0 0 0} is the \texttt{identity} matrix
\begin{verbatim}
  1 0 0
  0 1 0
  0 0 1
\end{verbatim}

More generally, if \texttt{A} is any \texttt{array}, then \texttt{DpA} is equivalent to \texttt{Dp,A}. For example, if \texttt{A} is the matrix
\begin{verbatim}
  1 2 3
  4 5 6
\end{verbatim}
then \texttt{3 5pA} is the matrix
\begin{verbatim}
  1 2 3 4 5
  6 1 2 3 4
  5 6 1 2 3
\end{verbatim}

The expressions \texttt{DpX} and \texttt{0 3pX} and \texttt{0 3pX} and \texttt{0 0pX} are all valid; any one or more of the dimensions of an \texttt{array} may be zero.

\textbf{Uses of empty arrays}. A \texttt{vector} of dimension zero contains no components and is called an \texttt{empty vector}. Three expressions which yield \texttt{empty vectors} are \texttt{1 0}, \texttt{""} and \texttt{.0} applied to any \texttt{scalar}. An \texttt{empty vector} prints as a blank line.

One important use of the \texttt{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 \texttt{N} in a base \texttt{B} \texttt{number} system shows a typical use of the \texttt{empty} \texttt{vector} in initializing a \texttt{vector} \texttt{Z} which is to be built up by successive \texttt{catenations}:
\begin{verbatim}
VZ=B BASE N
[1] Z=0
[2] Z=(B|B),Z
[3] N=(N-1)B
[4] Z=Z+0V
10 BASE 1776
1 7 7 6
8 BASE 1776
3 3 6 0
\end{verbatim}

Empty \texttt{arrays} of higher rank can be useful in analogous ways in conjunction with the \texttt{expand} 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 $X = \{2, 3, 5, 7, 11\}$ then $X[2]$ is $5$.

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 $X[\{3\}]$ 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 $\rho I$. 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:

$A$ = 'ABCDHG'  
$M = \begin{bmatrix} 3 & 0 & 3 & 1 & 4 & 2 & 1 & 4 & 1 & 2 & 4 & 1 & 4 \\ 3 & 2 & 1 & 4 \\ 4 & 1 & 2 \\ 4 & 1 & 4 \\ \end{bmatrix}$

$M = \begin{bmatrix} 3 & 2 & 1 & 4 \\ 4 & 1 & 2 \\ 4 & 1 & 4 \\ \end{bmatrix}$

$A = 'ABCDHG'$

$M = \begin{bmatrix} 3 & 2 & 1 & 4 \\ 4 & 1 & 2 \\ 4 & 1 & 4 \\ \end{bmatrix}$

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

$M = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ \end{bmatrix}$

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

$M = \begin{bmatrix} 2 & 3 & 4 \\ 10 & 11 & 12 \\ \end{bmatrix}$

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, and if $I$ is a matrix and $J$ is a scalar (or vice versa), $M[I; J]$ is a matrix.

The form $M[I; J]$ indicates that all columns are selected, and the form $M[I; J]$ indicates that all rows are selected. For example, $M[2; 1]$ is $5, 6, 7, 8$ and $M[2; 3]$ is $2, 1, 10, 9$.

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 $\rho P$. 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 $\rho P$ 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 $\rho P$ 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:

$X = [2, 3, 5, 7, 11]$  
$X[1, 3] = [6, 8]$  
$X = [2, 3, 5, 7, 11]$  

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.

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 of them comprises only a single element. 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, x is a scalar function, and P[M, N], then P[I,J] equals M[I,J] for I, J, and if Q=fX, then Q[I,J] is equal to f[M[I,J]].

If M and N are not of the same size, then P[M, N] 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 f[1]X[1]...f[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.

<table>
<thead>
<tr>
<th>Dyadic Function</th>
<th>Identity Element</th>
<th>Left-Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times</td>
<td>* 1</td>
<td>L R</td>
</tr>
<tr>
<td>Plus</td>
<td>+ 0</td>
<td>L R</td>
</tr>
<tr>
<td>Divide</td>
<td>/ 1</td>
<td>R</td>
</tr>
<tr>
<td>Minus</td>
<td>- 0</td>
<td>R</td>
</tr>
<tr>
<td>Power</td>
<td>** 1</td>
<td>R</td>
</tr>
<tr>
<td>Maximum</td>
<td>[1.7014E+38]</td>
<td>L R</td>
</tr>
<tr>
<td>Minimum</td>
<td>[1.7014E+38]</td>
<td>L R</td>
</tr>
<tr>
<td>Residue</td>
<td>0</td>
<td>L</td>
</tr>
<tr>
<td>Out of</td>
<td>1</td>
<td>L</td>
</tr>
<tr>
<td>Or</td>
<td>V 0</td>
<td>L R</td>
</tr>
<tr>
<td>And</td>
<td>A 1</td>
<td>L R</td>
</tr>
<tr>
<td>Equal</td>
<td>= 1</td>
<td>Apply</td>
</tr>
<tr>
<td>Not equal</td>
<td>≠ 0</td>
<td>for</td>
</tr>
<tr>
<td>Greater</td>
<td>&gt; 0</td>
<td>logical</td>
</tr>
<tr>
<td>Not less</td>
<td>≥ 1</td>
<td>arguments</td>
</tr>
<tr>
<td>Less</td>
<td>&lt; 0</td>
<td>only</td>
</tr>
<tr>
<td>Not greater</td>
<td>≤ 1</td>
<td>L</td>
</tr>
</tbody>
</table>

Table 3.5: IDENTITY ELEMENTS OF PRIMITIVE SCALAR DIADIC FUNCTIONS

For a matrix M, reduction can proceed along the first coordinate (denoted by f/M) or along the second coordinate (f/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).

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

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6
\end{pmatrix}
\]

then +/M is \(5 \ 7 \ 9\) and +/M is 6 15.
**Inner product.** The familiar matrix product is denoted by \(AB\). If \(A\) and \(B\) are matrices, then \(C\) is a matrix such that \(C[i,j]\) is equal to \(\sum A[i,k]B[k,j]\). A similar definition applies to \(Af.gB\) 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 \(C[i] = \sum A[k]/B[k,i]\). If \(B\) is a vector and \(A\) is a matrix, then \(C\) is a vector such that \(C[i] = \sum A[i,k]/B[g,k]\). If both \(A\) and \(B\) are vectors, then \(A\cdot B\) is the scalar product \(\sum A[k]/B[k]\).

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 \(\dim(A)\cdot\dim(B)\). (See the function drop in the section on Mixed Functions.) In other words, the dimension of the result is equal to \(\dim(A)\cdot\dim(B)\) for the two inner dimensions \(\dim(A)\) and \(\dim(B)\), which must agree and which are eliminated by the reduction over them.

Definitions for various cases are shown in Table 3.6.

<table>
<thead>
<tr>
<th>(pA)</th>
<th>(pB)</th>
<th>(pA\cdot pB)</th>
<th>Conformability requirements</th>
<th>Definition (Z = Af.gB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V)</td>
<td>(V)</td>
<td>(V)</td>
<td>(Z = Af.gB)</td>
<td>(Z = Tf.Agb)</td>
</tr>
<tr>
<td>(U)</td>
<td>(U)</td>
<td>(U)</td>
<td>(Z = Af.gB)</td>
<td>(Z = Tf.Agb)</td>
</tr>
<tr>
<td>(V)</td>
<td>(W)</td>
<td>(U)</td>
<td>(U = V)</td>
<td>(Z = Tf.Agb)</td>
</tr>
<tr>
<td>(T)</td>
<td>(U)</td>
<td>(T)</td>
<td>(T = U)</td>
<td>(Z = Tf.Agb)</td>
</tr>
<tr>
<td>(U)</td>
<td>(V)</td>
<td>(W)</td>
<td>(U = V)</td>
<td>(Z = Tf.Agb)</td>
</tr>
<tr>
<td>(T)</td>
<td>(U)</td>
<td>(T)</td>
<td>(T = U)</td>
<td>(Z = Tf.Agb)</td>
</tr>
</tbody>
</table>

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

**Outer product.** The outer product of two vectors \(X\) and \(Y\) with respect to a standard scalar dyadic function \(g\) is denoted by \(X\cdot Y\) and yields an array of dimension \((nx, ny)\), formed by applying \(g\) to every pair of components of \(X\) and \(Y\). If \(X\) and \(Y\) are vectors and \(Z = X\cdot gY\), then \(Z[i,j]\) is equal to \(X[i]Y[j]\). For example:

\[
\begin{align*}
X & = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \\
Y & = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \\
Z & = X\cdot Y \\
& = \begin{bmatrix} 4 & 5 & 6 \ 8 & 10 & 12 \ 16 & 20 & 24 \end{bmatrix}
\end{align*}
\]

Definitions for various cases are shown in Table 3.7.

<table>
<thead>
<tr>
<th>(pA)</th>
<th>(pB)</th>
<th>(pA\cdot.pB)</th>
<th>Definition (Z = Af.gB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(V)</td>
<td>(V)</td>
<td>(V)</td>
<td>(Z = AgB)</td>
</tr>
<tr>
<td>(U)</td>
<td>(U)</td>
<td>(U)</td>
<td>(Z = AgB)</td>
</tr>
<tr>
<td>(V)</td>
<td>(V)</td>
<td>(V)</td>
<td>(Z = AgB)</td>
</tr>
<tr>
<td>(W)</td>
<td>(W)</td>
<td>(W)</td>
<td>(Z = AgB)</td>
</tr>
<tr>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
<td>(Z = AgB)</td>
</tr>
</tbody>
</table>

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.

**Monadic transpose.** The expression \(\psi A\) yields the array \(A\) with the coordinates interchanged. For a vector \(V\) and a matrix \(M\), the following relations hold:

\[\psi V = V\]
\[\psi M = M^T\]

\(\psi V\) is equivalent to \(V\)
\(\psi M\) is the ordinary matrix transpose, that is, the rows of \(M\) are the columns of \(M^T\), and vice versa.
<table>
<thead>
<tr>
<th>Name</th>
<th>Sign</th>
<th>Definition or example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>$\rho A$</td>
<td>$\rho P \leftrightarrow 4$ $\rho E \leftrightarrow 3$ $\rho 5 \leftrightarrow 1$</td>
</tr>
<tr>
<td>Reshape</td>
<td>$V\rho A$</td>
<td>Reshape $A$ to dimension $V$ $3 , 4 \rho 1 2 \leftrightarrow E$ $12 \rho E \leftrightarrow 112$ $0 \rho E \leftrightarrow 10$ $\rho, 5 \leftrightarrow 1$</td>
</tr>
<tr>
<td>Ravel</td>
<td>$A$</td>
<td>$A \leftrightarrow (\times/\rho A) \rho A$ $\rho, 5 \leftrightarrow 1$</td>
</tr>
<tr>
<td>Catenate</td>
<td>$V, V$</td>
<td>$P, 12 \leftrightarrow 2 , 3 , 5 , 7 , 1 , 2 ,'T', , 'HIS' \leftrightarrow ,'THIS'$ $P[2] \leftrightarrow 3$ $P[4 , 3 , 2 , 1] \leftrightarrow 7 , 5 , 3 , 2$</td>
</tr>
<tr>
<td>Index</td>
<td>$[A]$</td>
<td>$E[1 , 3 , ; , 3 , 2 , 1] \leftrightarrow 3 , 2 , 1$ $E[1;] \leftrightarrow 1 , 2 , 3 , 4$ $E[;1] \leftrightarrow 1 , 5 , 9$ $'ABCDEFGHijkl'[E] \leftrightarrow EFGHijkl$</td>
</tr>
<tr>
<td>Index generator</td>
<td>$S$</td>
<td>First $S$ integers $14 \leftrightarrow 1 , 2 , 3 , 4$ $10 \leftrightarrow$ an empty vector</td>
</tr>
<tr>
<td>Index of</td>
<td>$V_1 A$</td>
<td>Least index of $A$ $P_1 3 \leftrightarrow 2$ $P_1 E \leftrightarrow 3 , 1 , 2 , 5$</td>
</tr>
<tr>
<td>Take†</td>
<td>$V^+ A$</td>
<td>Take or drop $</td>
</tr>
<tr>
<td>Drop†</td>
<td>$V^+ A$</td>
<td>Elements of coordinate $I$ $-2 + P \leftrightarrow 5 , 7$</td>
</tr>
<tr>
<td>Grade up †</td>
<td>$\Delta A$</td>
<td>The permutation which $\Delta 3 , 5 , 3 , 2 \leftrightarrow 4 , 1 , 3 , 2$ would order $A$ (ascending or descending) $\Delta 3 , 5 , 3 , 2 \leftrightarrow 2 , 1 , 3 , 4$</td>
</tr>
<tr>
<td>Grade down †</td>
<td>$\Phi A$</td>
<td>$10 , 1 , 0 , 0 / P \leftrightarrow 2 , 5 , 1 , 0 , 1 , 0 / E \leftrightarrow 5 , 7 , 9 , 11$ $1 , 2 , 3 , 4 \leftrightarrow 1 , 0 , 1 , 0 / E$ $9 , 10 , 11 , 12$</td>
</tr>
<tr>
<td>Compress</td>
<td>$V / A$</td>
<td>$D C B A$ $L K J I$ $I J K L$</td>
</tr>
<tr>
<td>Expand</td>
<td>$V \setminus A$</td>
<td>$1 , 0 , 1 , 0 / 2 \leftrightarrow 2 , 5 , 1 , 0 , 1 , 0 / 2 \leftrightarrow 5 , 7 , 9 , 11$ $1 , 0 , 1 , 1 , 1 , \setminus 2 \leftrightarrow E , F , G , H$ $I , J , K , L$</td>
</tr>
<tr>
<td>Reverse</td>
<td>$\phi A$</td>
<td>$\phi X \leftrightarrow HGFE$ $S X \leftrightarrow EFGH$</td>
</tr>
<tr>
<td>Rotate</td>
<td>$A \phi A$</td>
<td>$3 \phi P \leftrightarrow 7 , 2 , 3 , 5 \leftrightarrow \sim 1 \phi P$ $1 , 0 , \sim 1 \phi X \leftrightarrow EFGH$ $L , I , J , K$</td>
</tr>
<tr>
<td>Transpose</td>
<td>$V \phi A$</td>
<td>Coordinate $I$ of $A$ $2 , 1 \phi X \leftrightarrow BFJ$ becomes coordinate $CGK$ $V[I]$ of result $1 , 1 \phi E \leftrightarrow 1 , 6 , 11$ $DGHL$</td>
</tr>
<tr>
<td>Memberships</td>
<td>$A \epsilon A$</td>
<td>$\rho W \epsilon Y \leftrightarrow \rho W$ $E \epsilon P \leftrightarrow 1 , 0 , 1 , 0$ $0 , 1 , 1 , 0$</td>
</tr>
<tr>
<td>Decode</td>
<td>$V \setminus V$</td>
<td>$10 , 1 , 7 , 7 , 6 \leftrightarrow 1727$ $24 , 60 , 60 , 1 , 1 , 2 , 3 \leftrightarrow 3723$</td>
</tr>
<tr>
<td>Encode</td>
<td>$V \setminus S$</td>
<td>$24 , 60 , 60 , T , 3 , 7 , 23 \leftrightarrow 1 , 2 , 3$ $60 , 60 , T , 3 , 7 , 23 \leftrightarrow 2 , 3$</td>
</tr>
<tr>
<td>Deal †</td>
<td>$S ? S$</td>
<td>$W ? Y \leftrightarrow$ Random deal of $W$ elements from $Y$</td>
</tr>
</tbody>
</table>

Table 3.8: PRIMITIVE MIXED FUNCTIONS (see adjacent notes)
†. This function not available on APL/1130

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 $S[V]$ 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:
   - $P$:
     \[ P \leftrightarrow 2 \ 3 \ 5 \ 7 \]
   - $E$:
     \[ E \leftrightarrow 5 \ 6 \ 7 \ 8 \]
   - $X$:
     \[ X \leftrightarrow EFGH \]
   - $M$:
     \[ M \leftrightarrow 9 \ 10 \ 11 \ 12 \]
   - $A$:
     \[ A \leftrightarrow ABCD \]

Notes to Table 3.8

**Rotate.** If $X$ is a scalar or one-element vector and $Y$ is a vector, then $X \oplus Y$ is a cyclic rotation of $X$ defined as follows: $X \oplus Y$ is equal to $X[(1+(aX)\bmod{\dim{Y}})+1]$.

**Expand.** Expansion is the converse of compression and is denoted by $\cup \setminus Y$. If $Y=Y \setminus x$, then $Y/x$ is equal to $X$ if $X$ is a vector of numbers $\setminus Y/x$ is an array of zeros. In other words, $Y \setminus X$ expands $X$ to the form indicated by $U$, the elements of $X$ assuming the positions of the ones in $U$ and zeros filling in elsewhere. To be conformable, $+/U$ must equal $\mu X$.

**Decode.** The expression $R \oplus X$ denotes the value of the vector $X$ evaluated in a number system with radices $R[1], R[2], \ldots, R[k]$. For example, if $\mu R=24 \ 60 \ 60$ and $X=1 \ 2 \ 3$ is a vector of elapsed time in hours, minutes, and seconds, then $R \oplus X$ has the value $3723$, and is the corresponding elapsed time in seconds. Similarly, $W[X-1]$ is equal to $1776$, and $W[X-1]$ is equal to $1$. Formally, $R \oplus X$ is equal to $W[\mu X]$, whereas is the weighting vector determined as follows: $W[\mu X]$ is equal to $1$, and $W[X-1]$ 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, $3 \times 1 = 1776$. The arguments are not restricted to integer values. If $X$ is a scalar, then $X \cdot C$ 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 $R \cdot W$ denotes the representation of the scalar $R$ in the base-$W$ number system. Thus, if $R = 3 \cdot W \cdot N$, then $(x/R)_{N-W} = 2$ is equal to zero. For example, 2 2 2 5 5 is 0 1 0 1 and 2 2 3 5 is 1 0 0 1 0 1. The dimension of $R \cdot W$ 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 $J+V;S$ 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 $1+V$.

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

```
'ABCDEFGB' 'GAPPE'
7 1 6 6 5
```

If $X$ is a numerical vector, then the expression $X[I]/X$ 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 $7/X$ is 13 and $X[I]/X$ is 4.

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

**Membership.** The function $X \cdot Y$ yields a logical array of the same dimension as $Y$. Any particular element of $X \cdot Y$ 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, $(17) \cdot 3 5$ is equal to 0 0 1 0 1 0 0 and 'ABCDEFGB' 'GAPPE' equals 0 1 0 1 1 0 0.

If the vector $U$ represents the universal set in some finite universe of discourse, then $U \cdot A$ 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 $\epsilon$ is determined by the size of the left argument, whereas the size of the result of the dyadic function $\cdot$ 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 $N$, then $S+V$ takes the first $S$ components of $V$. For example, if $V=17$, then $3+V$ is 1 2 3 and $0+V$ is 10, and $8+V$ yields a domain error.

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

If $A$ is an array, then $W \cdot A$ 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 $A = 3 4 \cdot 12$, then $2 \cdot 3 \cdot A$ is the matrix

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

The function drop (\$) is defined analogously, except that the indicated number of elements are dropped rather than taken. For example, $-1 \cdot A$ 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 $k \cdot V$ produces the permutation which would order $V$, that is $V(k \cdot V)$ is in ascending order. For example, if $V$ is the vector 7 1 6 5 3 9, then $V$ is the vector 7 3 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 $\downarrow$ is formed by overstriking $I$ and $\wedge$.

If $P$ is a permutation vector, then $k \cdot P$ 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, $6 5 3 7 3 9 2$ is the vector 6 2 4 1 3 5.

# These functions not available on APL\1130
The right argument of \( \cdot \) 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 \( [j] \). The form \( \cdot A \) applies as usual to the last coordinate. The result of \( \cdot A \) 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 \( \downarrow (\{V\}) = \{V \} \) has the value \( 1 \) if and only if the elements of the vector \( V \) are all distinct.

Deal. The function \( \oplus N \) produces a vector of dimension \( N \) obtained by making \( N \) random selections, without replacement, from the population \( 1 \cdot N \). In particular, \( \oplus N \) yields a random permutation of order \( N \). Both arguments are limited to scalars or one-element arrays.

Comments. The lamp symbol \( \mathbf{a} \), formed by overstriking \( n \) and \( \mathbf{z} \), 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 \( (+) \) may (like any other function) occur repeatedly in a single statement. For example, the execution of the statement \( Z \leftarrow X \cdot A = 3 \) 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:

\[
X \leftarrow Y + 1 \quad Z \leftarrow 0
\]

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 \( J \leftarrow 0 \cdot I + 1 \) first augments \( I \), and then branches to statement \( S \) if \( N \) exceeds the new value of \( I \).

---

In the expression \( Z \leftarrow (A \cdot B) \cdot (C \cdot D) \) it is immaterial whether the left or the right argument of the \( \cdot \) 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 \( Z \leftarrow (A \cdot B) \cdot A \) is then executed. If the left argument of \( \cdot \) 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 5. 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 \( Z \).

# This function not available on APL\1130

91 92
Appendix A

SAMPLE TERMINAL SESSION

3×4  
X=3×4  
X  
7  
1=4×2"2  
P=1 2 3 4  
P×P  
5  
10 15 20  
Q='CATS'  
CATS  
Y2=5  
Y2+5  
Y2+Y21  
10  
+3+4×5+6  
+  
+5+6  
18  
X=3  
Y=4  
(X+Y)+4  
16  
X×Y+4  
24

xy
SYNTAX ERROR  
xy
^  
VALUE ERROR  
xy
^  
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

FUNDAMENTALS

Entry automatically indented  
Response not indented  
X is assigned value of  
the expression  
Value of X typed out  
Negative sign for negative  
constants

Exponential form of constant

Four-element vector  
Functions apply element by element

Scalar applies to all elements  
Character constant (4-element  
vector)

Multi-character names

Correction by backspace  
and linefeed

Executed from right to left

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

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 defn
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
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)

Display state indicator
Suspended on line 1 of BIN
Clear state indicator
Insert line to initialize Z
Execute revised function
Display revised function
and close definition

A multiplication drill
\(p^N\) 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 (Z) input
Make Z an empty vector
D is the length of Z
Append character keyboard entry
Branch to 3 if length increased
(i.e., entry was not empty)

Keyboard entries
Empty input to terminate
Display Q

Mixed output statement

RECTANGULAR ARRAYS

Dimension of P
Character vector
Catenation
$M = 2$
$3 p 2$
$3 5 7 11 13$

Display of an array of rank 2 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

A permutation vector

Permutation of P

A new permutation

FUNCTIONS ON ARRAYS

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

Sum (element-by-element)
Maximum

Comparison

Sum-reduction of V.

Product-reduction

Sum over first coordinate of M 
(down columns)

Sum over second coordinate of M 
(over rows)

Maximum over last coordinate

M+xN

Ordinary matrix (+,x inner) 
product

An inner product

+.,x inner product with vector 
right argument

MxN

7 9 4
5 8 6
9 8 7
M=N

0 0 0
0 0 1
1 1 0
+/V

10 
×/V

14

+M

13 22 12

+M

20 14 13

T/M

9 8 7

V

2 1 7

V=×15

2 4 6 8 10
1 2 3 4 5
7 14 21 28 35

V=×19

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

MIXED FUNCTIONS

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

Q=×10p5

Q

1 4 3 4 5 4 2 1 4 2

+Q=×15

2 2 1 4 1

Q=M

7 5 1
9 8 5
4 1 7
Rotate to left by 3 places
Rotate to right by 3 places
Rotate columns by different amounts
Rotation of rows all by 2 to right
Rotation of rows
Reversal of Q
Reversal of M along first coordinate
Reversal along last coordinate

Compression of Q by logical vector U
Compression by not U
Compression along first coordinate of M
Compression along last coordinate

Expansion of iota 3
Expansion of rows of M

Expansion of literal vector inserts spaces
Base 10 value of vector 1 7 7 6
Base 8 value of 1 7 7 6
4 digit base 10 representation of number 1776
3 digit base 10 representation of 1776

Mixed base value of 1 3 25 (time radix)
Representation of number 3805
Base 2 value
\[ M = \begin{bmatrix} 3 \end{bmatrix} \text{SP 'THREE SHORT WORDS'} \]

A matrix of characters

\[ \begin{array}{ccc}
J & \text{A} & \text{B} \\
20 & 8 & 18 & 5 & 5 \\
19 & 8 & 15 & 18 & 20 \\
23 & 15 & 18 & 4 & 19 \\
\end{array} \]

Ranking of \( M \) produces a matrix

\[ \text{INDEXING BY A MATRIX PRODUCES A MATRIX} \]

\[ \begin{array}{c}
\text{THREE SHORT WORDS} \\
\end{array} \]

Indexing by a matrix produces a matrix

\[ P = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 & 13 \end{bmatrix} \]

Index of 7 in vector \( P \)

7 is 4th element of \( P \)

6 does not occur in \( P \), hence result is 1 + \( P \)

A permutation vector

\[ Q = \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \]

\[ R = \begin{bmatrix} 2 & 4 & 3 & 5 \end{bmatrix} \]

\[ A = 'A'B'C'D'E'F'G'H'I'J'K'L'M'N'OP'Q' \]

A is the alphabet

Rank of letter C in alphabet is 3

\[ U = 'A'C'W'0'IS'T'H'E'N'O'S'T'W' \]

Membership

\[ \begin{array}{c}
U/A \\
000100110100110011001001 \\
E/H/IM/K/N/O/S/T/W' \\
(18) \times 3 7 5 \\
0 0 1 0 1 0 1 0 1 0 \\
\end{array} \]
BIBLIOGRAPHY

Berry, P.C., APL\130 Primer, IBM Corporation, 1968, Form No. C20-1697-0.


Smillie, K.W., Statpack 1: An APL Statistical Package, Publication No. 9, Department of Computing Science, University of Alberta, Edmonton, Canada, 1968.