AN INTRODUCTION
Hayden Computer Programming Series

COMPREHENSIVE STANDARD FORTRAN PROGRAMMING
James N. Haag

BASICS OF DIGITAL COMPUTER PROGRAMMING (Second Ed.)
John S. Murphy

BASIC BASIC: An Introduction to Computer Programming in BASIC Language (Second Ed.)
James S. Coan

ADVANCED BASIC: Applications and Problems
James S. Coan

DISCOVERING BASIC: A Problem Solving Approach
Robert E. Smith

ASSEMBLY LANGUAGE BASICS: An Annotated Program Book
Irving A. Dodes

PROGRAMMING PROVERBS
Henry F. Ledgard

PROGRAMMING PROVERBS FOR FORTRAN PROGRAMMERS
Henry F. Ledgard

FORTRAN WITH STYLE: Programming Proverbs
Henry F. Ledgard and Louis J. Chmura

COBOL WITH STYLE: Programming Proverbs
Louis J. Chmura and Henry F. Ledgard

SNOBOL: An Introduction to Programming
Peter R. Newstead

FORTRAN FUNDAMENTALS: A Short Course
Jack Stelinger

THE BASIC WORKBOOK: Creative Techniques for Beginning Programmers
Kenneth E. Schuman, Jr.

BASIC FROM THE GROUND UP
David E. Simon

APL: AN INTRODUCTION
Howard A. Peelle
AN INTRODUCTION

HOWARD A. PEELLE
University of Massachusetts
Amherst, Massachusetts

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey
Greetings! *APL: An Introduction* provides a set of *self-teaching* materials which provide an informal introduction to APL.* They are called “U-Programs” based on the belief that you will learn APL best if you program it yourself.

The APL U-Programs are designed for students at secondary and college levels who have a penchant for experimentation. Specifically, the materials may be used in the following ways:

1. Problem-Solving Exercises: Begin by observing examples of how APL functions and commands are used. Some examples show the computer’s results; others (marked with an arrow \(\Rightarrow\) in the margin) have the computer’s display omitted and are exercises for you to do. Answers are provided in the Appendix. (Note that this does not require access to a computer; if one is available, it can be used to enter problems, observe results, and check answers.)

2. Experimentation and Exploration: Using the problems (from 1.) as samples, explore the nature of APL functions and commands by conducting “experiments” on the computer. For instance, an experiment might involve systematically varying different values with the same function, or trying different functions or combinations of functions.

3. Formalization and Generalization: Using the results of experimentation (from 2.) as an intuitive basis, then formally express—either in words (to a human instructor) or in a program definition (to the computer)—general rules for describing the behavior of an APL function. Such “simulations” and other programs may be written similarly to apply APL for your own purposes.

In all of these ways, the user of APL U-Programs is encouraged to use a “heuristic” approach to learning APL. That is, by examining patterns in the examples shown and from results of experiments conducted, the student may make reasonable conjectures about the nature of the APL language. These conjectures may be confirmed by subsequent experience or by an instructor or a manual.

*APL is a Programming Language, which was developed by Kenneth E. Iverson of IBM Corporation. Originally conceived as a unifying mathematical notation in the late 1950s and early 1960s, APL has since been implemented on a variety of computing systems and has been used successfully in business, scientific research, and education. For a list of APL publications, write: APL Press, Box 378, Pleasantville, N.Y. 10570.
The APL U-Programs are organized into nine units—each with a title page/table of contents and review. The learning progression is designed to be sequential but may be altered by skipping forward or backward at the student’s discretion. Annotations in the right-hand margin are intended as supplementary explanation and may be overlooked by the independent-minded student.

Beginning with U-Program 1, APL tools for problem-solving are presented, and soon thereafter sample programs are demonstrated. Each of the U-Programs assumes a clear workspace, i.e., you enter expressions on an empty slate. APL expressions are indented 6 spaces, and the computer’s response is shown at the left margin. Some expressions on the page are simply examples to be observed. Other expressions are exercises for the student to do (here the computer’s response has been omitted and an arrow shown instead). Additionally, some expressions are marked “challenge” for those who wish to stretch their understanding.

At any rate, enjoy APL.

Howard A. Peelle

Amherst, Mass.
ABOUT APL SYSTEMS

The version of APL used in this book corresponds closely to the standard IBM program product, as implemented on the IBM 360 and 370 series time-sharing systems and the IBM 5100 desk-top computer. Other versions of APL, such as APLUM (APL at the University of Massachusetts) implemented on a CDC CYBER 74, differ slightly and would affect the following topics in this book:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significant Digits</td>
<td>3</td>
</tr>
<tr>
<td>Attention Key</td>
<td>56</td>
</tr>
<tr>
<td>E Notation</td>
<td>59</td>
</tr>
<tr>
<td>Deleting Lines in a Program</td>
<td>61</td>
</tr>
<tr>
<td>Trace Command</td>
<td>64</td>
</tr>
</tbody>
</table>

Note also that the user is expected to arrange access to an APL computing system and, therefore, that this book does not describe equipment, sign-on procedures, or any aspects pertaining to interaction with a particular machine. Rather, it assumes that one is ready to study the APL language, per se.
APL KEYBOARD
U-Program 1

COMMAND EXECUTION

Contents

Data Representation 2
Arithmetic Functions + − × ÷ 2
Error Reports 3
Names and the Assignment Command ← 4
Reassignment and Counters 5
Vectors 6
Parallel Processing 8
Catenation , 9
Relational Functions = < ≤ ≥ > ≠ 11
Maximum, Minimum, and Residue Functions ⌈ ⌊ ⌋ 14
Review 17
Problems 18
DATA REPRESENTATION AND ARITHMETIC FUNCTIONS

You type this

'HELLO'

HELLO

The computer responds here

'THese are examples of APL expressions'

You write the answer here (As if you were the computer)

'These are examples of APL expressions'

'With some for you to do'

Examples of the arithmetic functions

2 + 5

7

9 - 6

3

3 x 4

12

100 ÷ 5

20

4 + 8

You Do These

7 - 3

5 x 20

100 ÷ 4
Decimal numbers are written in the normal way

3.6 + 1.2
4.8

2.5 + 7.1

8 - 9
-1

4 - 7

6 × 2.00
12

3.0 × 5

100 ÷ 6
16.66666667
100 ÷ 3

100 ÷
SYNTAX ERROR
100 ÷
^ ERROR REPORTS

The computer reports an error--indicating that for proper syntax some number must follow ÷.
(^ points to the function nearest the error)

No harm has been done.
You may continue.

12 is equivalent to 12.00

10 significant digits are printed
NAMES AND THE ASSIGNMENT COMMAND

\[ A + 13 \rightarrow \text{Command to store some value and give it a name} \ (\text{Here 13 is the value; } A \text{ is the name}) \]

\[ A \rightarrow \text{type } A \text{ and the computer prints the value of } A \]

This can be read as "B is assigned to be 10"

\[ B + 10 \rightarrow \]

\[ B \]

\[ A + B \]

\[ 23 \]

\[ A - B \]

\[ \rightarrow \]

\[ A \times B \]

\[ (\text{the value of } A) \times (\text{the value of } B) \]

\[ \rightarrow \]

\[ A + B \]

\[ 1.3 \]

\[ B \times C \]

VALUE ERROR

\[ B \times C \]

\[ ^\wedge \]

Here the computer reports an error indicating that C does not (yet) have a value. If you do \[ B \leftarrow 10 \] first, \[ C \leftarrow 5 \] then \[ B \times C \] is OK.
REASSIGNMENT AND COUNTERS

Several letters may be used for a name, like YEAR
(Numbers and underscore — and Δ also may be used in names; but names
cannot have spaces nor begin with a number.)
determined by the latest assignment.

The value of a name is

\[ \text{COUNTER} + 1 \]
\[ \text{COUNTER} \]
1
\[ \text{COUNTER} + \text{COUNTER} + 1 \]
\[ \text{COUNTER} \]
2
\[ \text{COUNTER} + \text{COUNTER} + 1 \]
\[ \text{COUNTER} \]
3
\[ \text{COUNTER} + \text{COUNTER} + 1 \]
\[ \text{COUNTER} \]
→
\[ \text{COUNTER} + \text{COUNTER} + 1 \]
\[ \text{COUNTER} \]
→
\[ \text{YEAR} \]
→

\[ \text{COUNTER} \text{ is } 1 \]

\[ \text{COUNTER becomes } 1 \text{ plus the old value of COUNTER} \]

\[ \text{COUNTER is increased by } 1 \text{ again} \]

\[ \text{COUNTER is increased by } 1 \text{ to become ?} \]

\[ \text{What is the value of COUNTER now?} \]

A name keeps its value
(unti it is re-assigned)
VECTORS

SET is initially assigned 2

SET is reassigned. It becomes the value of SET with 3 chained on the end (chains values together)

SET is reassigned to be 2 3 with 5 chained on the end

SET is 2 3 5 with 7 chained on

SET is 2 3 5 7 11

SET is now a set of numbers (called a "vector") and can be treated as a single entity.

For example, 1 is added to each element of SET (2-1), (3-1), (5-1), (7-1)

2 times each element of SET
Assigning two vectors (with spaces between the numbers)

Element-by-element multiplication

\[(2+6), (3+6), (5+6), (7+6), (11+6)\]

(An equivalent way of adding 6)

chains the values together
PARALLEL PROCESSING

V and W are assigned values
(Notice that each vector has 5 values)

Element-by-element addition
(This is "parallel processing")
Element-by-element subtraction

\[
\begin{aligned}
V + W & = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 4 & 0 & 1 & 5 & 3 \end{bmatrix} \\
6 & 3 & 6 & 12 & 14 \\
-2 & 3 & 4 & 2 & 8 \\
\end{aligned}
\]

\[
\begin{aligned}
V \times W & = \begin{bmatrix} 2 \\ 6 \\ -2 \\
3 \\ 3 \\
5 \\
7 \\
11 \\
14 \\
\end{bmatrix} \\
& = \begin{bmatrix} 2 \\
18 \\
-2 \\
9 \\
15 \\
35 \\
77 \\
154 \end{bmatrix}
\end{aligned}
\]

Element-by-element multiplication

\[
\begin{aligned}
V \cdot W & = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 4 & 0 & 1 & 5 & 3 \end{bmatrix} \\
& = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 4 & 0 & 1 & 5 & 3 \end{bmatrix}
\end{aligned}
\]

Element-by-element division

\[
\begin{aligned}
V \cdot W & = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 4 & 0 & 1 & 5 & 3 \end{bmatrix} \\
& = \begin{bmatrix} 2 & 3 & 5 & 7 & 11 \\ 4 & 0 & 1 & 5 & 3 \end{bmatrix}
\end{aligned}
\]
CATENATION,
is the "catenation" function. You have already seen it used with numbers (page 6).
The catenation function can be used with literal data too:

```
D + '*'       \ D is assigned the literal *
D
*
D . D         \ chains the value of D together with itself
**
D . D . D . D
```

```
E + 'A'
S + 'A*A'
I + '00'
G + 'A'
N + '0*'


*ΔΔ*Δ00Δ0*

```

This is a literal vector

chain the symbols together (in the order shown)
Storing letters (literals) in names

E = 'HINGT'
G = 'WAS'
O = 'ON'

G, E, O

WASHINGTON

A = 'ABRA'
L = 'CAD'
A, L, A

# = 'SE YOU, RE'
O = 'D BAR'
T = 'ON!!!'
S = 'CUR'

S, H, O, T

Note: These are "overstrike" symbols (Type ' backspace .)
RELATIONAL FUNCTIONS = < ≤ ≥ > ≥

The Equals Function

Does 5 equal 5?
1 means "yes" (true)

0 means "no" (false)

Not true

\{ \}

For you to do

= with a vector

5 is compared with each element of the vector

1s result where there are 5s and 0s result everywhere else
The Less-Than Function

Is 3 less than 5?
1 (true)

Is 8 less than 5?
0 (false)

Is 7 less than 5?
0 (false)

Is 6 less than 5?
0 (false)

Is 5 less than 5?
1 (true)

The Less-Than or Equal Function

The Greater-Than Function

The Greater-Than or Equal Function

Answer 1 if true and 0 if false

The Not-Equals Function
8 \times 9  
1

8 \times 8  
0

4 \times 4  

4 \times 7  

'A' = 'B'
1

'C' = 'C'
0

'O' = 'O'

'F' = 'L'

'B' = 'ABBABA'

'B' = 'ABBABA'

8 \text{ is not equal to } 9 \text{ (true)}

8 \neq 8 \text{ is false (0)}

Not-Equals and Equals can be used with literals too.
A new function symbol $\Gamma$ for you to experiment with:

Use two numbers, one on each side.

Try them on opposite sides (commuted)

Try several experiments at once (using a vector)

$\Gamma$ yields the larger of all corresponding elements

$\Gamma$ is the Maximum Function
Experiments with L

What does L do?

What would you call this function?

How does L relate to F?
Challenge:

Here's a challenge!
Figure out how I works.

3 | 8
0 1 2 3 4 5 6 7
0 1 2 0 1 2 0 1
3 | 9 10 11

4 | 4 5 6 7 8 9 10 11 12

5 | 5 10 15 20 40 -5
0 0 0 0 0 0
4 | -3 -2 -1 0 1 2 3
1 2 3 0 1 2 3
5 | -6 -4 -2 0 2 4 6

7 6 5 4 3 2 1 | 14
0 2 4 2 2 0 0
REVIEW

In APL, there are two types of data which can be represented: literal data, and numerical data. Literal data are represented with enclosing quote marks; numbers are written in the usual way, with decimal points and negative symbols where appropriate.

A variety of functions exists in the APL language, perhaps the most fundamental of which are the arithmetic functions and catenation. In addition to these and the relational functions (a family of functions which compare data and result only in 0s or 1s), the maximum, minimum, and residue functions, there are many more to be explored!

All expressions in APL are composed of data and/or functions (usually both together) and can be executed immediately by the computer. This is called the "command execution" mode.

Names may be used—at your discretion—to store data in the computer. The assignment command is used whenever a name is created or whenever data stored in a name are changed. One particularly useful application of this is the use of a counter to keep track of a value which is increased at certain times.

Vectors are linear collections of data and may be either literal or numerical. Through the use of vectors, a single function may be applied to many elements simultaneously. This "parallel processing" is convenient and often useful for experimentation.

Error reports occur whenever you have asked the computer to execute some expression it "doesn't understand." At that point, you may simply retype your expression or enter a new expression. In any event you are not penalized. You may continue as if nothing had happened.

To test your understanding of this first U-Program, try the problems on the next page and check your answers on the computer.
PROBLEMS

\[ T + 3.2 \ 6 \]
\[ S + 4 \ -2 \]

\[ 'T + S' \]

\[ T + S \]
\[ T - S \]
\[ T \times S \]
\[ T \div S \]

\[ T . S \]
\[ T = S \]
\[ T < S \]
\[ T > S \]
\[ T \leq S \]
\[ T \geq S \]
\[ T \neq S \]

\[ T \mid S \]
\[ T \perp S \]
\[ T \perp S \]
# U-Program 2

## PROGRAM DEFINITION

### Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining a Program</td>
<td>20</td>
</tr>
<tr>
<td>Executing a Program</td>
<td>20</td>
</tr>
<tr>
<td>Editing a Program</td>
<td>21</td>
</tr>
<tr>
<td>&quot;Monadic&quot; vs. &quot;Dyadic&quot; Functions</td>
<td>25</td>
</tr>
<tr>
<td>The Iota Function $\iota$</td>
<td>25</td>
</tr>
<tr>
<td>Indexing [ ]</td>
<td>26</td>
</tr>
<tr>
<td>The Rho Function $\rho$</td>
<td>28</td>
</tr>
<tr>
<td>Sum-Reduction $+/$</td>
<td>30</td>
</tr>
<tr>
<td>A Monadic Program</td>
<td>31</td>
</tr>
<tr>
<td>$\textsc{ERASE}$ — A System Command</td>
<td>32</td>
</tr>
<tr>
<td>Review</td>
<td>33</td>
</tr>
<tr>
<td>Problems</td>
<td>34</td>
</tr>
</tbody>
</table>
DEFINING A PROGRAM

This is a program to compute the area of a square.

A del ▽ is used to begin program definition and is followed by a name. (This program is named AREA.) Each expression is entered on successive lines of the program, [1], [2], etc.

A second del ▽ closes program definition.

EXECUTING A PROGRAM

To execute this program, type its name. (AREA)

Then each line is performed by the computer. (Note that this program required a value for S.)

Program AREA can be executed again, perhaps for a different value for S.

Execute program AREA for $S \leftarrow 9$

If $S$ is assigned several values, executing AREA produces results for all values simultaneously.
EDITING A PROGRAM

\[ \text{AREA}[1] \ 'THE AREAS ARE' \]  \[ \text{Changing line [1]} \]

\[ \text{AREA}[\Box] \] \[ \text{Command to display the (latest) program definition} \]

\[ \text{AREA} \] \[ \text{Note that line [1] has been changed} \]

\[ 1 \times 1 \times S \]

\[ S + 1 2 3 4 5 \]

\[ \text{AREA} \]

\[ \text{THE AREAS ARE} \]

\[ 1 \ 4 \ 9 \ 16 \ 25 \]

\[ S + 1 2 3 4 5 \]

\[ \text{AREA} \]

\[ \text{You Execute AREA now} \]

\[ \rightarrow \]
Additional programs may be defined too.

For example, here is another program, named BASEBALL.

```
\begin{verbatim}
\text{BASEBALL}
[1] 'THIS PROGRAM COMPUTES BATTING AVERAGE.'
[2] H \div AB
[3] V

V \leftarrow V + H + 61
AB \leftarrow AB + 200
\text{BASEBALL}
\end{verbatim}
```

Note that \( H \) and \( AB \) must be assigned first, before executing BASEBALL.

```
\text{THIS PROGRAM COMPUTES BATTING AVERAGE.}
0.305
```

Execute BASEBALL for these values of \( H \) and \( AB \):}

```
H \leftarrow H + 63
AB \leftarrow AB + 200
\text{BASEBALL}
```


This is another program, named TRIANGLE, which computes the area of a triangle, given its base (B) and height (H). The result will be stored in A.

There is no value for A until the program is executed.

Executing TRIANGLE does not cause anything to be printed—although the program did do something! Typing A produces the result.

Execute TRIANGLE for the values of B and H assigned. What is A?
command to add line(s) to program TRIANGLE

The computer prints [3] for you. You type the expression A. Then, after [4], type a \( \checkmark \) to end.
The new definition looks like this.

Now line [3] will print the value of A.

Executing TRIANGLE now prints the area of a triangle with base 6 and height 14.

Execute TRIANGLE for these 5 bases and 5 heights.

Note: When you define a new program, be sure to give it a different name.
"MONADIC" VS. "DYADIC" FUNCTIONS

THE IOTA FUNCTION

\( \iota \) is a function which uses only one number -- written on its right.

\( \iota \) is a "monadic" function whereas \( + - \times \div \) have been shown as "dyadic" functions -- with numbers written on the left and right.

\( \iota \) returns a vector of positive integers up to and including the integer given (see also p. 95)

This is called the "null" vector (a blank line)

Take a guess at these...
INDEXING [ ]

$V + 2 3 5 7$  
$V$ is a vector of four elements

$V[1]$  
The first element in $V$

$V[2]$  
The second element in $V$

$V[3]$  
The third element in $V$

$V[4]$  
The fourth element in $V$

$V[5]$  

$\rho$  
counts the number of elements
$W + 5\ 9\ 2\ 0\ 7\ 1$

$W[1]$

5

$W[2]$

The 2nd element in $W$

$W[3]$

The 3rd element in $W$

$W[2\ 3]$

The 2nd and 3rd elements (an index can be a vector)

$W[2 + 3]$

An index can be the result of an expression


Add the 2nd and 3rd elements

$W[4]$

0

$W[5.5]$

The index must be an integer!

$W[6]$

$W[7]$

INDEX ERROR

$W[7]$

An index cannot be > the total number of elements in the vector.
THE RHO FUNCTION $\rho$

$X + \ -7\ -6\ -5\ -4\ -3\ -2\ -1$

$\rho X$

There are 7 elements in vector $X$.

$Y + 6\ 6\ 6\ 6\ 6\ 6\ 6$

$\rho Y$

How many elements in $Y$?

$\text{ALPHABET} + 'ABCD$EGHIJKLMNOPQRSTUVWXYZ'$

$\rho \text{ALPHABET}$

26 letters in ALPHABET

$\rho 'ABCD'$

$\rho 'A\ C\ E'$

$\rho$ Counts spaces too.

$\text{SHAKESPEARE} + 'A MIDSUMMER NIGHT$'S DREAM'$

$\rho \text{SHAKESPEARE}$

Count the literals

TITLE + 'A MIDSUMMER NIGHT$'S DREAM'$

TITLE

A MIDSUMMER NIGHT$'S DREAM$

$\rho \text{TITLE}$

25
L + 'TRIAL'

\( pL \)

5

L[4] The fourth element of L is 'A'

A

L[1] The first element of L is ?

\[ L[5\ 3\ 4\ 2] \]

\( L[IAR] \)

\[ L[2\ 4\ 1]\; \text{', '},\; L[1\ 4\ 3\ 5]\; \text{', '} \]

\( \text{Chain these together} \)

L[5] + 'D'

\( L[1] \)

\( \text{TRIAD} \)

\( \text{Now L is} \)

L[2 4 5] + 'WST'

\( L[1] \)

\( \text{What is L after these reassignments?} \)

pL How many elements in L?
SUM-REDUCTION +/

V + 2 3 5 7

+/V

17

2 + 3 + 5 + 7

17

W + 5 9 2 0 7 1

+/W

24

5 + 9 + 2 + 0 + 7 + 1

→

+/\\ 10

55

+/\\ 19

→

+/W adds up the elements in W

V is a vector

+/ adds up the elements of a vector

It is equivalent to placing + signs between the elements and evaluating the result.

The sum of the integers from 1 to 10

Add up the integers from 1 to 9
A MONADIC PROGRAM

DEFINING A PROGRAM TO FIND AVERAGES:

\[
\begin{align*}
Y & = 4 \; 8 \; 9 \\
\text{SUM} & = +/Y \\
N & = \rho Y \\
\text{SUM} \div N & \\
\end{align*}
\]

Some numbers named \( Y \)
The sum of the elements in \( Y \)
The number of elements in \( Y \)
The average of the elements in \( Y \)

\[
\begin{align*}
\text{\textsc{vaverage}} \; X \\
[1] \; \text{SUM} & = +/X \\
[2] \; N & = \rho X \\
[3] \; \text{SUM} \div N \\
[4] \; \text{v} \\
\end{align*}
\]

A program to find the average of any vector of numbers \( (X) \).

Note that the name of the program is \textsc{average} and that \( X \) stands for the numbers the program will use. To execute this program some numbers must appear to the right of \textsc{average}. \textsc{average} is, therefore, a "monadic" program.

\[
\begin{align*}
\text{\textsc{average}} & \; 4 \; 8 \; 9 \\
7 \\
\end{align*}
\]

These numbers are the values.

7 is the result.

\[
\begin{align*}
\text{SUM} & \\
N & \\
\text{\textsc{average}} & \\
\text{syntax error} & \\
\text{\textsc{average}} & \wedge \\
\end{align*}
\]

\( \text{SUM} \) is the sum of the elements in \( X \)
(from line [1])

\( N \) is the number of elements in \( X \)
(from line [2])

\textsc{average} requires numbers for \( X \)
For values assigned to \( V \), the program prints the AVERAGE of the numbers in \( V \).

\( W + 5 9 2 0 7 1 \)

AVERAGE \( W \)

\( \rightarrow \)

\( W[4] + 6 \)

\( W \)

\( \rightarrow \)

AVERAGE \( W \)

\( \rightarrow \)

\( \)ERASE — A SYSTEM COMMAND

\( \)ERASE AVERAGE

AVERAGE \( W \)

SYNTAX ERROR

AVERAGE \( W \)

\( \)^

This is a "system command" which will erase any name.

Now the program AVERAGE is erased

(and a new AVERAGE program could be defined)

**Challenge:**

Rewrite program AREA as a "monadic" program.

Rewrite program BASEBALL as a "dyadic" program.
The mode of APL in which programs are defined is called "program definition" mode (or "function definition" mode). Here expressions may be entered—one line at a time—for execution later by the computer. The sequence of expressions is given a name so that when execution is desired, you need only use that name.

Program definition begins and ends with a del \( \forall \) symbol. (In fact, dels should always be paired, since they act to switch back and forth from command execution mode to program definition mode.) Each expression entered is preceded by a line number, and programs—once defined—may be edited by referring to line numbers. For example, lines may be replaced, or new lines may be added. Only the latest version of an edited program is stored by the computer.

Several important points about programs are:

—More than one program may be defined at a given time, but each must have a different name.

—Programs may use values from (global) names assigned either outside or inside a program.

—Results of executed programs may or may not be displayed, depending on their definition.

Some additional primitive functions, helpful in defining programs, are: iota \( \iota \), rho \( \rho \), indexing \([ ]\), and sum-reduction \(+./\). Iota generates index integers beginning with 1 (unless directed otherwise); rho computes the size of numerical or literal data (e.g., the number of elements in a vector); indexing is used to select individual elements of data (numerical or literal); and sum-reduction adds up numbers. Iota and rho are examples of "monadic" functions.

All monadic functions in APL appear with data only on the right hand side. They differ from "dyadic" functions, such as \(+ - \times \div\), which are used with data on both sides. From syntax alone, then, you can distinguish between monadic and dyadic functions.

Programs may be defined to be monadic or dyadic or nyladic (the latter meaning no input data).

To test your understanding of U-Program 2, execute (by hand) the program on the next page. Check your results against the computer's.
\[
S + 9 \\
V + 2 3 5 7 11 13 \\
L + 'AEHNRSTW' \\
\]

**REVIEW**

[1] \[L[7 3 2 9 1 4 6 8 2 5 6 9 1 5 2]\]
[2] \[\rho L\]
[3] \[\rho V\]
[4] \[\backslash S\]
[7] \[+/V[5-2 3]\]
[8] \[+/V\]
[9] \[+/\backslash S\]
[10] \[V[\rho V]\]
[11] \[V[\backslash \rho V]\]
[12] \[+/V[\backslash \rho V]\]
[13] \[+/V[\rho V]-S\]
[14] \[+/\backslash V[3]+\rho V\]

\[\backslash\]

**REVIEW**
U-Program 3

EVALUATING EXPRESSIONS

Contents

Rules for Evaluating Expressions 36
Generalized Reduction 39
Review 44
Problems 45
**RULES FOR EVALUATING EXPRESSIONS**

When two (or more) functions occur in one expression, the rules for evaluation are:

1. **Rule 1:** The function inside parentheses is done first.

2. **Rule 2:** Inside the parentheses (or when there are none) the rightmost function is done first.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$5 + 4 \times 2$</td>
<td>13</td>
</tr>
<tr>
<td>$5 + (4 \times 2)$</td>
<td>13</td>
</tr>
<tr>
<td>$(5 + 4) \times 2$</td>
<td>18</td>
</tr>
<tr>
<td>$(10 \times 3) + 4$</td>
<td>34</td>
</tr>
<tr>
<td>$10 \times (3 + 4)$</td>
<td>70</td>
</tr>
<tr>
<td>$10 \times 3 + 4$</td>
<td>70</td>
</tr>
<tr>
<td>$(6 \times 4) + 5$</td>
<td>36</td>
</tr>
<tr>
<td>$6 \times (4 + 5)$</td>
<td>60</td>
</tr>
<tr>
<td>$6 \times 4 + 5$</td>
<td>29</td>
</tr>
<tr>
<td>$6 + 4 \times 5$</td>
<td>29</td>
</tr>
<tr>
<td>$6 + (4 \times 5)$</td>
<td>29</td>
</tr>
</tbody>
</table>
(2 \times 3) + (4 \times 5)

(2 \times 3) + 4 \times 5

2 \times 3 + 4 \times 5

(first) \quad 4 \times 5

then \quad 3 + 20

then \quad 2 \times 23

1 + 10 \times 9 - 2

\{ \text{These are equivalent expressions} \}

\{ \begin{align*}
1 + (10 \times (9 - 2))
\end{align*} \}

(2 \times 3 + 5 \times 4) = (2 \times (3 + (5 \times 4)))

Are these two expressions equal?

\rightarrow

\textbf{Challenge:}

\begin{align*}
21 & + 3 \times 8 \div 3 + 4 \div 2 \\
22 & + 3 \times (8 \div (5 + (4 \div 2))) \\
21 & = 22
\end{align*}

1

21

\rightarrow
A Long expression

Evaluation by pieces:

\[ 16 + 5 \mid 4 \times -3 \div 2 \]
\[ T + -3 \div 2 \]
\[ T \]
\[ S + 4 \times T \]
\[ S \]
\[ R + 5 \mid S \]
\[ R \]
\[ Q + 6 + R \]
\[ Q \]
\[ P + 1Q \]
\[ P \]

Evaluating expressions in APL is different from the way it is done in algebra (\(\times\) and \(\div\) first, then \(+\) and \(-\)).

One reason for this is that there are so many functions in APL (about 60 in all!) that it would be hard to remember which to do first.
This is how sum-reduction is actually evaluated:

\[ W = 5 \ 9 \ 2 \ 6 \ 7 \ 1 \]

\[ +/W \]

30

\[ 5 + 9 + 2 + 6 + 7 + 1 \]

\[ \rightarrow \]

\[ 5 + (9 + (2 + (6 + (7 + 1)))) \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 1 \]

\[ \text{SUM} \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 7 \ + \ \text{SUM} \]

\[ \text{SUM} \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 6 \ + \ \text{SUM} \]

\[ \text{SUM} \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 2 \ + \ \text{SUM} \]

\[ \text{SUM} \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 9 \ + \ \text{SUM} \]

\[ \text{SUM} \]

\[ \rightarrow \]

\[ \text{SUM} \ + \ 5 \ + \ \text{SUM} \]

\[ \text{SUM} ! \]
Some other dyadic functions may be used with the reduction symbol `/.

**Times-reduction**

```
x/ is evaluated like
+
```

(only with $x$ in place of $+$)

**Maximum-reduction**

```
/ is evaluated similarly
```

(This is the largest value in the vector)
Minimum-reduction
$L/W$
also evaluates from "right to left"

(This is the smallest value in the vector)

Minus-reduction
Note that the result here is not the same as the algebraic sum of the numbers.
The rightmost operation is done first,
then the next rightmost,
then the next rightmost,
and so on
until the last operation is completed.

$W + 5 9 2 6 7 1$
$\rightarrow$
$MIN + 7 \ L \ 1$
$MIN + 6 \ L \ MIN$
$MIN + 2 \ L \ MIN$
$MIN + 9 \ L \ MIN$
$MIN + 5 \ L \ MIN$
$MIN$
$\rightarrow$
$-W$
$\rightarrow$
$DIFF + 7 \ - \ 1$
$DIFF$
$DIFF + 6 \ - \ DIFF$
$DIFF$
$DIFF + 2 \ - \ DIFF$
$DIFF$
$DIFF + 9 \ - \ DIFF$
$DIFF$
$DIFF + 5 \ - \ DIFF$
$DIFF$

EVALUATING EXPRESSIONS

41
minus-reduction of $1 2 3 4 5 6$

step by step

---

Challenge:

$S + 16$

$(+/S[1 3 5]) - +/S[2 4 6]$

---

Reduction is generalized for use with dyadic functions $+ - \times \div$

$\Gamma \ L \ \text{and} \ \lfloor \ \text{etc.}$
Check yourself on the rules for evaluating expressions.

\[ 2 \times 14 \]

\[ 14 \times 2 \]

\[ (14) \times 2 \]

\[ 3 + 2 \times 14 \]

\[ \frac{3}{3} + 2 \times 14 \]

\[ 32 \]

\[ \text{CENTIGRADE} + 20 + 10 \times -1 + 14 \]

\[ \text{CENTIGRADE} \]

\[ \text{FAHRENHEIT} + 32 + 9 \times \text{CENTIGRADE} \div 5 \]

\[ \text{FAHRENHEIT} \]
REVIEW

APL has simple rules for evaluating expressions—even those containing many different functions. Basically, the rule is to evaluate the function on the far right—subject to parentheses, which dominate in the normal way—and repeat until the entire expression is done. There is no hierarchy of functions in APL (as there is in conventional algebra).

Another way of viewing the APL rule for evaluating expressions is the following: Every function—dyadic or monadic—uses the entire expression on its right. (This amounts to evaluating the rightmost function first.)

Reduction operation is generalized to apply to many dyadic functions, including $+$, $-$, $\times$, $\div$, $\int$, $\l$, and $\l$. The $/$ notation is always preceded by a dyadic function which is (effectively) inserted between elements of the data following, and then the resulting expression is evaluated.

Test your understanding of these APL rules by evaluating the expressions on the next page.
PROBLEMS

\[
\begin{align*}
R & = 5 \ 5 \ 10 \ 4 \ 5 \ 20 \\
E & = \frac{3}{2} \\
V & = 2 \ 3 \ 5 \ 7 \ 11 \ 13 \\
I & = 2 \\
E & = 4 \\
W & = 5 \ 9 \ 2 \ 6 \ 7 \ 1 \\
\end{align*}
\]

\[\therefore E \]

\[\therefore I \times E \]

\[\therefore E \times I \]

\[\therefore (E) \times I \]

\[\therefore E + I \times E \]

\[\therefore +/E + I \times E \]

\[\therefore E + I \times E - I \]

\[\therefore (E + I) \times E - I \]

\[\therefore (E + I) \times (E - I) \]

\[\therefore +/V \times W \]

\[\therefore -/R < W \]

\[\therefore (\frac{1}{W}) - \frac{1}{W} \]

\[\therefore (+/W) \div \frac{1}{W} \]

\[\therefore R \ | \ E \ | \ V \ | \ I \ | \ E \ | \ W \]

\[\therefore \frac{1}{R}, \ E, \ V, \ I, \ E, \ W \]
U-Program 4

BRANCHING

Contents

Logical Functions \(\wedge\) \(\vee\) \(\sim\) 48
Compression \(/\) 52
Iteration and Counters 55
Unconditional Branching \(\rightarrow\) 56
Editing: Inserting New Lines 57
Conditional Branching 58
Header Editing 60
More Editing: Deleting Lines 61
Programs with Explicit Results 62
The Trace Command 64
The Power Function \(\ast\) 65
The Absolute Value Function \(|\cdot|\) 68
The Floor and Ceiling Functions \(|\cdot|\) 71
Review 73
Problems 74
LOGICAL FUNCTIONS

\[ L + 1 \quad 1 \quad 0 \quad 0 \]
\[ K + 1 \quad 0 \quad 1 \quad 0 \]

\[ L \land K \]
\[ 1 \quad 0 \quad 0 \quad 0 \]
\[ 1 \land 1 \]
\[ 1 \land 0 \]
\[ 0 \land 1 \]
\[ 0 \land 0 \]

\[ L \lor K \]
\[ 1 \quad 1 \quad 1 \quad 0 \]
\[ 1 \lor 1 \]
\[ 1 \lor 0 \]
\[ 0 \lor 1 \]
\[ 0 \lor 0 \]

\[ \neg L \]
\[ 0 \quad 0 \quad 1 \quad 1 \]
\[ \neg 1 \]
\[ \neg 0 \]

The **AND function** \( \land \)

The result is 1 for both 1 and 1; 0 otherwise.

The **OR function** \( \lor \)

The result is 1 if one or the other (or both) is 1; 0 otherwise.

The **NOT function** \( \neg \)

The result is the logical opposite.

The **LOGICAL functions** (AND, OR, NOT) only operate on logical data (0s and 1s).
\[ \neg L \lor K \]

0 0 0 1

\[ \neg L \land K \]

"NOT" (L "OR" K)

\[ (\neg L) \land \neg K \]

"NOT" (L "AND" K)

\[ \rightarrow \]

\[ (\neg L) \land \neg K \]

( "NOT" L ) "AND" ( "NOT" K )

**Challenge:**

\[ +\neg(L \land \neg K) \land L \lor \neg L = K \]

\[ \rightarrow \]
And-reduction

\[ \land / 1 \ 1 \ 0 \ 0 \] is equivalent to \[ 1 \land 1 \land 0 \land 0 \]

\[ \land / L \]
\[ 1 \ 0 \ 0 \ 0 \]
\[ \land / K \]
\[ 1 \ 0 \ 1 \ 0 \]

\[ \land / L = L \]
\[ 1 \]

\[ \land / K \]
\[ 0 \]

\[ \land / L \neq L \]
\[ 1 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / L = L \]
\[ 1 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]

\[ \land / K \]
\[ 0 \]
\( \land \lor K \)

Does \( L \lor K \) result in all 1s?

\( \lor \land K \)

Does \( L \land K \) result in any 1s?
Values of $K$ and $L$ are chained together and named $Q$

$$Q + K, L$$

$$Q$$

10101100

$$P + 2 \ 3 \ 5 \ 7 \ 11 \ 13 \ 17 \ 19$$

$$P$$

235711131719

$$Q \times P$$

$$Q \div P$$

251113

$$(\sim Q) \div P$$

371719

The COMPRESSION function /

This expression may be read as:
"$Q$ compress $P$".

Only those elements in $P$ which have corresponding 1s in $Q$ appear in the result:

Here the result is elements in $P$ where 1s appear on the left:

$$\downarrow \downarrow \downarrow \downarrow \downarrow$$

01010011

235711131719

Note: The COMPRESSION function requires a logical vector of 0s and 1s on the left -- one for each element on the right.
COMPRESSION works by:
- keeping values where there are 1s
- omitting values where there are 0s

1 1 0 0
6 2 8 4
(keep)(keep)(omit)(omit)

Keep the values where there are 1s in K

COMPRESSION works similarly with literals

1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 / 'STOP THE RECORD'
Keep the 3 and the 5

Keep the 3; omit the 5

Omit the 3; keep the 5

Omit the 3 and the 5

1 / 6

0 / 6

a blank line (the null vector)

Compression of a single element either returns that element -- as in 1/6 or returns the null vector -- as in 0/6

This fact is used in branching.
ITERATION AND COUNTERS

Program POW computes and prints four powers of $N$ by repeated multiplication

$\text{POW}$

\begin{verbatim}
[1] Z + 1
[2] Z + Z \times N
[3] Z
[4] Z + Z \times N
[5] Z
[6] Z + Z \times N
[7] Z
[8] Z + Z \times N
[9] Z \text{ on end of program definition with a } \checkmark
\end{verbatim}

Note: You may end

$N + 3$

$\text{POW}$

\begin{verbatim}
3  ----- caused by line [3]
9  ----- " " " [5]
27 ----- " " " [7]
81 ----- " " " [9]
\end{verbatim}

$N \leftarrow 4$

$\text{POW}$

\rightarrow

Execute $\text{POW}$ for $N \leftarrow 4$
Program POWOW accomplishes what program POW does (and more) by iteration; that is, it repeats certain statements by using an unconditional branch command.

Line [4] → 2 can be read as: "go to line [2]" and causes the program to repeat lines [2] and [3] (indefinitely)....

This is known as an "endless loop"--a programmer's nightmare!

Push ATTN key (on top right of keyboard) to stop the computer printing
POWOW should be edited so that it will stop after a certain number of repetitions.

This command inserts a new line between lines [1] and [2] (you are then given the opportunity on [1.6] to insert more lines.)

Line [1.6] may be overridden to insert a new line between lines [3] and [4]. (Again, [3.6] invites you to insert more lines.)

Overriding with a new line [4] replaces what was on line [4].

Now, displaying the whole program POWOW includes the new lines.

Also, note that the lines have been automatically renumbered.
**CONDITIONAL BRANCHING**

\[ \text{VPOWOW}() \]

V POOW

1. \( Z + 1 \)
2. \( I + 0 \)
3. \( Z + Z \times N \)
4. \( Z \)
5. \( I + I + 1 \)
6. \( (I < 9)/3 \)

\[ \text{N} \]

POOW

I is initialized as 0.

I is incremented by 1 (each iteration.)

Line [6] is a **conditional branch command**

it can be read as:

"branch to line [3] if I is less than 9 -- otherwise, go to the next line"

The general format is

\[ \rightarrow (\text{condition})/\text{line number} \]

(Note that branching works this way because of the compression function.)

Now, the program stops after printing 9 powers of N.
Editing `POWOW` again so that it will stop after a certain number of iterations.


X is a name used by the program and therefore must be assigned a value.

Here X is 12 (iterations)

Note that large numbers -- here, larger than 10 billion -- displayed in E notation (similar to "scientific notation") where E may be read as "...times ten to the..." The same holds for very small numbers -- like one billionth is 1E-9.

X can be easily changed (as can N)

You execute it.
HEADER EDITING

Program POWOW may be changed so that the values for N and X can be entered at the same time as the program name.

This editing command is used to change line [0] (the "header" of the program)

or, even the name of the program can be changed.

Now POWER is a new "dyadic" program

It uses two values:
one on the left (for N)
one on the right (for X)

Notice that each value for Z is printed (due to line [4]).
MORE EDITING: DELETING LINES

This is the procedure for deleting a line: cause the computer to type the line number and then press ATTN, followed immediately by RETURN. That line will be deleted; and all lines affected in the program will be renumbered after the final del D.

If line [4] is deleted, program POWER will not print anything...

although the final result can be obtained by typing Z.
PROGRAMS WITH EXPLICIT RESULTS

If the header of a program assigns a value, it has an "explicit result".

POWER is changed to have an explicit result (Z).

When POWER is executed, now whatever value Z has at the end of the program is printed as the result.
Only the final value of Z is printed.
The program itself has this result. The importance of this is that the program can now be used in expressions.
(see page 68)

Execute POWER for an N of 2 and an X of 3
VPOWER([]) V

V Z + N POWER X

[1] Z + 1
[2] I + 0
[3] Z + Z × N
[4] I + I + 1
[5] + (I < X) / 3

V

The display of POWER

Z is local to the program
(as are N and X); they are only placeholders for values the program will use when executed. They do not keep their values outside of the program. (See also p. 81)

When POWER is executed, the result is only temporarily assigned to Z.

As soon as the program terminates, Z returns to its previously assigned value (see p. 61)

Similarly for N and X (see p. 59)
THE TRACE COMMAND

Command to “trace” lines 1 thru 5 of program POWER

When the program is traced, all results of execution -- for each line indicated -- are printed out.

Program POWER simulates the behavior of the primitive function * for positive integers on the right.
THE POWER FUNCTION *

3 \* 2
9

4 \* 2
16

6 \* 2
→

5 \* 2
7 \* 2
9 \* 2
25 49 81

5 \* 15
5 25 125 625 3125

7 8 9 10 \* 1 2 3 4

3 \* 3
27

3 \* 3 \* 3
→

3 \* 4
→

3 \* 3 \* 3 \* 3
→

3 \* 5
243

Several numbers to the power 2
5 to several powers
Several numbers to several
different powers

3 "to the power" 3
(3 cubed)

3 "to the power" 4

3 to the 5th power
Square roots (numbers to the $\frac{1}{2}$ power)

Cube root

Fifth root

Fractional powers are permitted

Negative powers ($2^{-1}$ or $\frac{1}{2}$)

An APL curiosity
(Take a guess)
Three different absolute value programs ("absolute value" is the positive value of a number):

1. **ABSOLUTEVALUE using *:

```
VABSOLUTEVALUE
[1] (X * 2) * .5V
X + 8 -8

ABSOLUTEVALUE
```

2. **Absolute value using branching**

```
VAB X
[1] + (X < 0) / 4 — if X is a negative number, go to 4
[2] X — otherwise, print X and then
[3] +0 — stop
[4] -X V — print the negative of X

AB 8
```

Note — 0 (or any other line number not in the program) causes it to stop immediately.

3. **ABSolute value using explicit result.**

```
VZ + ABS X
[1] Z + X — assume X is a positive number and make that number the result (Z)
[2] +(X > 0) / 0 — if X is positive, stop
[3] Z + -X V — otherwise, the result (Z) is changed to the negative of X

ABS -11
```

11

```
ABS 11
```
THE ABSOLUTE VALUE FUNCTION

\[ |-8| = 8 \]

\[ |8.88| = 8.88 \]

\[ |-3 \times -3| = 9 \]

\[ |3 \times -3| = 9 \]

\[ |5 \times -8| = 40 \]

\[ \text{ABS } 5 \times -8 \]

\[ T + \text{ABS } -340 \div 17 \]

\[ T \]

\[ T + AB -340 \div 17 \]

\[ \text{SYNTAX ERROR} \]

\[ T + AB -340 \div 17 \]

\[ \wedge \]

The defined function ABS performs identically to \(|\) and can be used in expressions ... whereas program AB cannot (because it has no explicit result)
Defined program RES models the residue function (see p.16)
(see also p.200 for the formal definition of .)

\[ VR + A \text{ RES } B \]

1. \( \rightarrow (A = 0) \land B < 0 \) / 0
   - if \( A \) is 0 and \( B \) is negative, stop (residue is not defined)
2. \( R + B \)
3. \( \rightarrow (A = 0) / 0 \)
   - if \( A \) is 0, branch to 0 (stop); the result is \( B \)
4. \( R + R - |A| \)
   - if \( R \) is positive, subtract absolute values of \( A \) repeatedly
5. \( \rightarrow (R \geq 0) / 4 \)
6. \( R + R + |A| \)
   - if \( R \) is negative, add absolute values of \( A \) repeatedly until \( R \) is nonnegative.
7. \( \rightarrow (R < 0) / 6 \)

5 RES 13

\( \rightarrow \)

You try it for 5 and 13

\( T \Delta \text{RES } + 17 \)

5 RES 13

RES[1]
RES[2] 13
RES[3]
RES[4] 8
RES[5] 4
RES[4] 3
RES[5] 4
RES[5]
RES[6] 3
RES[7] 3

\( T \Delta \text{RES } + 0 \)

Removing the trace
5 RES 13

5 | 13

1 RES 3.14
0.14

3.14 - 1 RES 3.14

This yields the fractional part RES can be used in an expression ...

... just like the | function.

Challenge:

\[ v D \leftarrow \text{FLOOR} \ N \]

\[ v \ D \leftarrow \ N - 1 \lfloor N \rceil \]

\[ \text{FLOOR} \ 3.14 \]

\[ v S \leftarrow \text{CEILING} \ N \]

\[ v \ S \leftarrow \ N + 1 \lfloor -N \rceil \]

\[ \text{CEILING} \ 3.14 \]
THE FLOOR AND CEILING FUNCTIONS

The FLOOR Function \( l \)

Floor yields the nearest integer down the number line. \((\leq)\)

The CEILING Function \( \Gamma \)

Ceiling yields the nearest integer up the number line. \((\geq)\)

An example of using \( \Gamma \) in indexing

(it assures you that the indices are integers)

SHERLOCK + 'SCOTLAND YARD'

\(H + 7 4 11 21 5 12\)

SHERLOCK[\(\lfloor 35 \times H \rceil + / H \)]

LONDON
A program to ROUND off numbers to the nearest integer

\[ VQ + \text{ROUND } N \]

[1] \[ Q + \ln + 0.5 \] \[ V \]

\[ \text{ROUND 3.14} \]

\[ \text{ROUND 3.6} \]

\[ \text{ROUND -2.55} \]

\[ \text{ROUND -2.0904} \]

Challenge:

\[ \boxed{[1] + X + 10 \div 6} \]

1.666666667

\[ (10 \times -3) \times 0.5 + X \times 10 \times 3 \]

1.667

\[ (10 \times -4) \times 0.5 + X \times 10 \times 4 \]

\[ \boxed{\text{displays the result of the expression to its right.}} \]
REVIEW

Branching is an important programming technique. It permits you to indicate the sequence of commands to be executed in a program. Both unconditional and conditional branching can be expressed. A popular form of conditional branch commands is \( \rightarrow \) (condition) / (line #) although any expression which evaluates to an integer or null may follow the branch symbol \( \rightarrow \).

The logical functions \( \land \) (AND), \( \lor \) (OR), \( \lnot \) (NOT), and the compression function / are often used within branch commands. Also, the power function \( \ast \), absolute value \( \mid \) (monadic), residue \( \mid \) (dyadic), ceiling \( \lceil \), and floor \( \lfloor \) are useful mathematical tools available on the keyboard. It should be noted that some APL functions serve double duty—that is, they can be used monadically or dyadically.

Additional program editing procedures include inserting new lines between existing lines, deleting lines, and changing the header of a program.

Programs with “explicit results” can be used within expressions just as if they were primitive functions on the keyboard. You may define such programs with local names—names which have values only while the program is being executed. The complete execution of a program (or any specified lines) may be traced automatically by the computer.

Test your understanding of branching and related functions with the exercises on the following page.
PROBLEMS

0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 / 'BEFORE YOU VIEW MORE,'

\[(4 = 4) \land 5 = 5\]

\[(3 \geq 4) \lor 5 \neq 5\]

\[\text{LOGICAL} = 0 1 1 0 0 1\]

\[(\sim \land / \text{LOGICAL}) = \lor / \sim \text{LOGICAL}\]

\[2 \mid + / \text{LOGICAL}\]

\[2 \ast + / \text{LOGICAL}\]

\[\mid + / \text{LOGICAL}\]

\[\mid - / \text{LOGICAL}\]

\[[] + S + 3\]

\[\Box + T + (S = tS + 1) / tS + 1\]

\[T \ast S\]

\[S \ast T\]

\[(S \ast 2) \mid 2 \ast S\]

\[((S + 1) \ast 2) = (S \ast 2) + (2 \ast S) + 1\]

\[(S \times \times / S - T) \ast .5\]

74  BRANCHING
\[ P + P = 2 \]
\[ X + 20 \div 3 \]
\[ (10 \times -P) \times 1.5 + X \times 10 \times P \]

Embodify the above expression in a program (with an explicit result) which will round-off a number X to P places.

Examine the program below

\[ V2 + L \text{ MAX } R \]
[1] \( (L > R) \div 4 \)
[2] \( Z + R \)
[3] \( Z + 0 \)
[4] \( Z + L \)

and then write a similar program (with branching) to find the MINimum of two numbers L and R.

Then,

\[ T_{\text{MIN}} + 14 \]
\[ R + 1.667 \text{ MIN } 2 \]
\[ T_{\text{MIN}} + 0 \]
\[ S \text{ MIN } R \text{ MAX } T [S] \]
U-Program 5

APPLYING FUNCTIONS

Contents

Random Number Generator ? 78
Local Names 81
The Membership Function ε 82
Sub-Programs 84
The Take Function † 85
The Drop Function ‡ 86
The Deal Function ? 88
The Grade-Up Function ¾ 91
The Grade-Down Function ½ 92
The Index-of Function ⊙ 93
Index Origin—A System Variable § 95
Review 96
RANDOM NUMBER GENERATOR

?2
2

?2
1

?2
1

?2


?3 3 3 3 3 3 3 3
1 1 2 3 1 2 1 3

?6 6 6 6 6
6 4 5 4 3

?52

(52) = 52
0

A random number from 1 to 52
(picked out of 52)

You may not get the same number
if you execute ?52 twice

If N is some integer between 1 and 52,
then N is ≥ 1
and
N is ≤ 52
and
N is a whole number
(remainder after dividing by 1 is 0)

Random integers from 1 to 3

See...?
Simulating the roll of two dice

Two random numbers, each between 1 and 6

A program to ROLL two "dice" and add them up.

(ROLL is a "nylaic" program with an explicit result Z.)

Lucky!

"Roll" two dice and add the numbers

Again — (it may be a different result)
Randomly picking a letter from the ALPHABET

A random index (1 to 26)

4 random indices
LOCAL NAMES

\( V \) RANDOMWORDS \( N \); \( J \)

[1] \( J + 0 \) — Initially \( J \) is 0

[2] ALPHABET[26 26 26 26] — 4 random letters are printed

[3] \( J + J + 1 \) — \( J \) is incremented by 1

[4] \( + (J < N) / 2 \) — branch to line 2 if \( J < N \) otherwise end the program

N is a local name.

Additional local names may be listed after semicolons in the program header.

\( J \) is a local name used by the program RANDOMWORDS (in addition to \( N \))

RANDOMWORDS 6

QPXL
KJDG
ROHJ
RSCK
XBVE
CSXY

\( J \)

VALUE ERROR

\( J \)

\( \wedge \)

\( N \)

29

Note that \( J \) has no value outside the program

\( \) RANDOMWORDS prints

\( N \) randomly generated

4-letter "words"

\( J \) is used to count (inside the program) the random "words"

And \( N \) is not changed

(This is the value of \( N \) from page 78 )
THE MEMBERSHIP FUNCTION $\in$

<table>
<thead>
<tr>
<th>$\in$</th>
<th>4 ∈ 2 4 6 8</th>
<th>Is 4 a member of 2 4 6 8? (Yes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 ∈ 2 4 6 8</td>
<td>Is 5 a member of 2 4 6 8? (No)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 ∈ 1 5</td>
<td>Is 6 a member of 1 2 3 4 5?</td>
<td></td>
</tr>
<tr>
<td>2 ∈ 1 5</td>
<td>Is 2 a member of 1 2 3 4 5?</td>
<td></td>
</tr>
</tbody>
</table>

VOWELS + 'AEIOU'

<table>
<thead>
<tr>
<th>∈</th>
<th>'A' ∈ VOWELS</th>
<th>Is 'A' a member of 'AEIOU'? (Yes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∈</th>
<th>'B' ∈ VOWELS</th>
<th>Is 'B' a vowel?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∈</th>
<th>'CAT' ∈ VOWELS</th>
<th>3 questions: Is 'C' a vowel? (no) Is 'A' a vowel? (yes) Is 'T' a vowel? (no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 0</td>
<td>give 8 answers (0s and 1s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∈</th>
<th>'COMPUTER' ∈ VOWELS</th>
<th>There is at least one vowel in 'COMPUTER'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

82 APPLYING FUNCTIONS
\[
\forall 'LINGO' \in \text{VOWELS} \\
\forall Z + \text{VOWELCHECKER \ WORD} \\
[1] Z + \forall \ \text{WORD} \in \text{VOWELS} \ \\
\text{VOWELCHECKER 'CONSONANTS'} \\
\text{VOWELCHECKER 'WHYZZ'}
\]

\[
\forall 0 1 0 0 1
\]

A program to check if a WORD has a VOWEL

Is there a vowel in 'CONSONANTS'? (yes or no)

0 or 1?
SUB-PROGRAMS

Programs may be used within other programs. They are called "sub-programs".

Program VOWELCHECKER is used within program RW as a sub-program.

It checks WORD (a random 4-letter "word" assigned on line [1]) for a vowel. If it doesn't have a vowel (0 = VOWELCHECKER WORD) the program branches back to line [1] to pick another word.

If it does have a vowel, the next line [3] prints the word before going back to line [1] again.

Uh oh. This program has no way of stopping.

You might edit RW to count the words printed out and to stop when it reaches a certain number (like RANDOMWORDS on page 81).

For now, use the ATTN key to stop it.
THE TAKE FUNCTION

1. \(3 + W\)
2. \(2 \text{ "take" } W\)
   (take the first 2 from the front of \(W\))
3. \(3 \text{ "take" } W\)
4. \(4 \text{ "take" } W\)

\(W = 6 + W\)

W is compared to 6 \(\text{ "take" } W\)

When taking more than the total number of elements, 0s are used (or spaces, with literal arrays).

Take 2 from the rear of \(W\)

Take the last 3 of \(W\)

\((2 + W), \ -4 + W\)

The last 5 of \(W\)

\((-5 + W\)

\((-8 + W\)

(Guess)
**THE DROP FUNCTION**

\[ W = 5 9 2 6 7 1 \]

1. \[ 2 + W \]
   \[ = 2671 \]

\[ 3 + W \]

2. \[ (4 + W) = 2 + W \]
   \[ = 11 \]

\[ (-2 + W) = 4 + W \]
   \[ = 1111 \]

3. \[ -4 + W \]

\[ -6 + W \]

4. \[ -3 \} 'APLOMB' \]
   \[ APL \]

5. \[ 3 \} 'APLOMB' \]
A program using DROP ↓

\[\text{VTRI } N\]

[1] \[N\] —— print the value of \(N\)

[2] \[N + 1 + N\] —— \(N\) becomes 1 "drop" \(N\)

[3] \[+(0 < \rho N)/1\] —— branch to line 1 if \(0 < \rho N\)

\(\text{TRI 'CHEAT'}\)

CHEAT
HEAT
EAT
AT
T

\(\text{Program TRI prints a triangle-shape of whatever you give it for } N\)
\(\text{(here 'CHEAT')}\)

\(\text{TRI 'ANYTHING'}\)

\(\text{TRIangle 'ANYTHING'}\)
THE DEAL FUNCTION?

5 numbers are randomly selected from 1 2 3 4 5 without replacement (the numbers are scrambled)

There are no repeats.

"Deal" the integers 15 at random.

'Stone'[I + 5 ? 5]

Scrambled letters by indexing

NOTES

What value for I was required for the above result?

1 "deal" 5
(1 random number from 15)

2 random numbers from 15

3 random numbers from 15

4 random numbers from 15

5 random numbers from 15
A simulated deal of a deck of cards

52 ? 52
43 47 13 7 49 14 25 26 12 19 30 17 48 44 36 35 22 20 18 2 4
40 33 31 9 28 37 39 1 52 6 42 11 10 27 46 41 45 51 38
8 29 16 5 34 15 3 23 32 50 24 21

(A bridge "hand")

13 ? 52
25 45 43 41 30 48 40 5 17 42 44 10 29

(A different "hand")

13 ? 52

13 "dealt" out of 213

13 ? 13

14 "dealt" out of 213 (guess)
Challenge:

\[ 'NOTES'[:I[I \pm 5 \ ? \ 5]] \]

What must I have been for the above to happen?

Repeated indexing producing permutations

It comes back to itself!
THE GRADE-UP FUNCTION

I is a "permutation vector" which happens to sort D1 into ascending order.

The grade-up function can be used to produce the same result.

Δ yields a permutation vector---which will arrange a vector in order

The ascending order of D2

A concise program to sort any numerical vector into ascending order

SORT scrambled V
THE GRADE-DOWN FUNCTION 🔄

overstrike ▽ and 1

produces a permutation vector for descending order

Arrange D2 in descending order

What are the indices which will arrange these in descending order?

N + 7 ? 7
S + 'NEPTUNE' [N]
S

Suppose N is some permutation vector

S is 'NEPTUNE' scrambled

S can be unscrambled by using grade-up 4.

Try it again.

N + 7 ? 7
S + 'NEPTUNE' [N]
S [△N]
THE INDEX-OF FUNCTION

ALPHABET

ABCDEFGHIJKLMNOPQRSTUVWXYZ

ALPHABET \ 'A' \ 1

ALPHABET \ 'BAT' \ 2 1 20

ALPHABET \ 'MAN' \ 18 15 2 9 14

ALPHABET \ 'ROBIN' \ 18 15 2 9 14

ALPHABET[18 15 2 9 14]

ALPHABET \ '' \ 27

1 + \(\rho\)ALPHABET

20 16 12 8 8 6 8

2 (used dyadically) yields the index-of 'A' in ALPHABET

'B' is the 2nd letter in ALPHABET
'A' is the 1st letter in ALPHABET
'T' is the 20th letter in ALPHABET

What are the indices of 'M' and 'A' and 'N' in ALPHABET?

If you use the indices with ALPHABET, you get ... ?

In case the value on the right is not found in the values on the left, index-of gives 1 + the number of values on the left.

In case of duplicates, index-of only gives the first index.
VZ + LSORT X

[1] Z + X [\ ALPHABET \ X] v

LSORT 'CAT'

ACT

LSORT 'SLOT'

Program LSORT will sort literals into alphabetic order.
INDEX ORIGIN — A SYSTEM VARIABLE \( \square IO \)

This command changes the origin of indices from 1 to 0.

It affects the deal function, the index-generator, the index-of function, as well as all indexing operations (see also: pp. 129, 131, chapter 8) and grade-up and grade-down.

The command to change the origin back to 1.

Normal execution

Note: \( \square IO \) is one of several "system variables" all of which begin with \( \square \). See 214.
REVIEW

APL has a rich resource of functions—you may consider them "tools"—to apply in programming. The random number generator $\, \sim \, $, for example, is convenient for simulating real-world processes, experiments, and, of course, games. Other useful functions include: membership $\ell \in \, $, take $\downarrow \, $, drop $\uparrow \, $, deal $\sim \, $, grade-up $\oplus \, $, grade-down $\ominus \, $, and index-of $\downarrow \, $.

Programs may have several local names (in addition to those needed for syntax). They are listed following semicolons in the program’s header and are used to keep track of values—such as in counters—which are only needed while the program is being executed. In fact, all local names lose their values after execution is completed.

Programs may have sub-programs. (And sub-programs may have sub-programs, etc.) They may be used within expressions, but only with the proper syntax. A main program continues execution exactly where it left off after a sub-program is completed.

Some system variables affect certain APL functions. $\Box IO$ is one such system variable which acts to change the index origin.

To test your understanding of U-Program 5, begin writing programs of your own choosing using these tools. See U-Program 6 for examples.
# U-Program 6

## INTERACTIVE PROGRAMS

### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Input: Numerical</td>
<td>98</td>
</tr>
<tr>
<td>Data Input: Literal</td>
<td>100</td>
</tr>
<tr>
<td>An Arithmetic Drill-and-Practice Program</td>
<td>102</td>
</tr>
<tr>
<td>Line Editing</td>
<td>104</td>
</tr>
<tr>
<td>Line Labels</td>
<td>104</td>
</tr>
<tr>
<td>Suspended Program</td>
<td>111</td>
</tr>
<tr>
<td>State Indicator</td>
<td>112</td>
</tr>
<tr>
<td>A Simple Game Program</td>
<td>114</td>
</tr>
<tr>
<td>A Simple Simulation Program</td>
<td>116</td>
</tr>
<tr>
<td>Review</td>
<td>118</td>
</tr>
</tbody>
</table>
DATA INPUT: NUMERICAL

$3 \times \square$  
□ (the "quad" symbol) requests input. The computer prints □: and then waits.

□: 

7

You must enter a number before $3 \times \square$ can be evaluated

$A + \square$  
Requesting input for A

□: 

5  
You enter 5

A  
A is now 5

$B + \square$  
Requesting input for B

□: 

9  
9 is entered

B

What is the value of B?

$C + 4 \times \square$  
C is 4 times the "quad" (to be some number)

□: 

8  
8 is entered for the quad

C  
Then C is 32

$N + \square$  
N is to be some thing

□: 

17 \times 2  
Evaluate this.

N  
N is that number
If you enter an improper expression (one which produces an error report), the request for input is repeated.

If you enter a right-pointing arrow \( \rightarrow \) (an empty branch), the request is terminated.

And \( S \) will not have a value.

**Challenge:**

5 × 8 \( \rightarrow \) \( \Box \) + 2

\( \Box \):

10

60

5 × 8 \( \rightarrow \) \( \Box \) + 2

\( \Box \):

7

With 10 in place of \( \Box \), the expression evaluates to be 60.

What is the evaluation of the expression for 7 in place of \( \Box \)?
DATA INPUT: LITERAL □

Overstrike □ and ' to form □

□ (quote-quad) requests literal input

The keyboard opens at the left margin. LITERALS are entered.

A now has the value 'LITERALS'

There are 8 elements (letters) in A

□ requests literal input

What is in B?

B has 5 elements

'ANY' is entered for C

What did you expect?

B , C , A

Enter any literals (spaces in between)
$X + 'TYLRENE'$

$X + [], X$

'ACE' goes where [] is

Then, what is the value of X?

$Y + []$

This "satisfies" the request for literal input... but Y does not have a value.
AN ARITHMETIC DRILL-AND-PRACTICE PROGRAM

VDRILL

[1] 'MULTIPLY'
[3] X
[5] Y
[6] ANSWER + ⊗
[7] +(ANSWER = X × Y) / 1
[8] 'NO, TRY AGAIN'
[9] → 6 V

DRILL

MULTIPLY

16
11
⊗:
176

MULTIPLY

14
13
⊗:
143
NO, TRY AGAIN
⊗:
272
NO, TRY AGAIN
⊗:
182
There's a problem with this program:

- the student has no way to stop it!

(If he gets a problem correct, he is given another one;
if he makes a mistake, he is given another try.
This will go on and on . . .)

unless → is entered

Let's insert a line which permits the student to stop the program.

STOP is assigned some number not likely to be a response to these multiplication problems

When the student enters STOP, the program is terminated.
The current definition of DRILL

```
VDRILL[]V
V DRILL
[1] 'MULTIPLY'
[3] X
[5] Y
[6] ANSWER+[]
[7] ->(ANSWER=STOP)/0
[8] ->(ANSWER=XxY)/1
[9] 'NO, TRY AGAIN'
[10] ->6
V
```

**LINE EDITING**

-- allowing you to change single characters on a line

```
VDRILL[8][21]
[8] ->(ANSWER=XxY)/1
[8] ->(ANSWER=XxY)/NEWPROB
[9] [167]
[1] 'MULTIPLY'
[8] NEWPROB:'MULTIPLY'
[2] [67]
[6] ANSWER+[]
[6] GUESS:ANSWER+[]
[7] [10] ->GUESSV
```

**LINE LABELS**

NEWPROB and GUESS are line labels.

Line labels are names followed by a colon and an expression on a line in a program.

They take on the value of the line number and may be used for convenience in branching.

The general form of line editing is

\[ \text{\_view} \text{ DRILL } \text{NEWPROB:'MULTIPLY'} \text{ GUESS:ANSWER+[] \_view} \]

\[ \text{ program name [n \_m] } \]

where \( n \) \equiv line number

and \( m \) \equiv number of spaces from

left margin to position the type ball

The new definition of DRILL

```
VDRILL[]V
V DRILL
[1] NEWPROB:'MULTIPLY'
[3] X
[5] Y
[6] GUESS:ANSWER+[]
[7] ->(ANSWER=STOP)/0
[8] ->(ANSWER=XxY)/NEWPROB
[9] 'NO, TRY AGAIN'
[10] ->GUESS
V
```
Line labels are **local to** the program;
that is, they have no values after completion of the program.

You may, therefore, safely use the same line labels in different programs without interference.
Refinements to DRILL

Now we will change DRILL to display multiplication problems in a different way.

VDRILL[1]\ display line [1]
[1] NEWPROB:'MULTIPLY'
[2] (push ATTN key)
[6] (All other lines remain the same)

VDRILL[1]V

\ DRILL
[1] NEWPROB:' ;X+?99
[4] GUESS:ANSWER+[]
[5] →(ANSWER=STOP)/0
[6] →(ANSWER=X×Y)/NEWPROB
[7] 'NO, TRY AGAIN'
[8] →GUESS

This is what DRILL looks like now

; is used for mixed output -- that is, when you want to print out numerical and literal data on the same line. (See lines [1] and [2] of DRILL)

Multiplication problems in a new format

42
\ 30
\ 1260

74
\ 89
\ STOP
Further refinements to DRILL

Editing DRILL to keep track of the total number of problems completed correctly (N) and the number of consecutive wrong answers (W).

```plaintext
VDRILL
[0] DRILL
[0] DRILL;N;W
[1] [.5] N+0
[0.6] NEWPROB;W+0
[0.7] N+N+1
[0.8] →(N>5)/END
[0.9] [1]6
[1] NEWPROB; 'X+?99
////////
[1] 'X+99
[7.6] [8][8]
[8] →GUESS
6
[8] →(W≤3)/GUESS
[9] 'LATER. GET SOME HELP NOW'
[10] →0
[11] END; 'THAT''S ALL.' V
```

The revised program

```plaintext
VDRILL
V DRILL;N;W
[1] N+0
[2] NEWPROB;W+0
[3] N+N+1
[4] →(N>5)/END
[7] 4p;
[8] GUESS;ANSWER+/
[9] →(ANSWER=STOP)/0
[10] →(ANSWER=X×Y)/NEWPROB
[11] 'NO, TRY AGAIN'
[12] W+W+1
[13] →(W≤3)/GUESS
[14] 'LATER. GET SOME HELP NOW'
[15] →0
[16] END; 'THAT''S ALL.'
```

initialize N to be 0
initialize W to be 0
increment N by 1
increment W by 1
branch to the END (line [16])
branch to GUESS (line [8])
after 5 correct problems
after 3 or fewer repeated wrong answers

display the problem
the student's ANSWER
if wrong, increment W by 1
(go to line [14] after the 4th wrong answer)
Execution of DRILL

DRILL

53
\_ 22
\_ 1166
\_ 20
\_ 93
\_ 1860
\_ 17
\_ 71
\_ 177
\_ NO, TRY AGAIN
\_ 1207
\_ 18
\_ 41
\_ 738
\_ 60
\_ 50
\_ 3000
\_ THAT'S ALL.

Terminates automatically after 5 problems are completed correctly.

DRILL

57
\_ 53
\_ 1551
\_ NO, TRY AGAIN
\_ 2521
\_ NO, TRY AGAIN
\_ 2831
\_ NO, TRY AGAIN
\_ 2921
\_ NO, TRY AGAIN
\_ LATER. GET SOME HELP NOW!

Terminates automatically after 4 consecutive wrong answers to a problem.

END

Does line label END have a value now?
a common editing procedure: display a line and then change it.

With these changes,

VDRILL[16[]]
[16] END:'THAT'S ALL.'
[16] END:'CONGRATULATIONS! WOULD YOU LIKE 5 MORE?'
[17] 'ENTER Y FOR YES, N FOR NO.'
[18] +(Y'ε+)/1
[19] 'O.K. NO HARD FEELINGS. SEE YOU NEXT TIME.'

DRILL

You execute the program

--- after getting 5 right, try entering Y or YES or ANYTHING
When you line edit here, e is elided, then you type in the ES and =.

Hey, You

Cheat!!

Obviously, "flaw" there's a "flaw" in this design... (I could be used to handle this.)

This error suspends the program (LENGTH ERROR results because the length (size) of 'YES' is not the same as 'No'.)

This State Indicator command indicates that program DRILL is suspended on line [18].
**SUSPENDED PROGRAM**

When a program is suspended, its execution has been halted before completion.

The values of local names are available,

and line labels too.

Most importantly, a suspended program may be resumed later—perhaps after correcting error(s)—by typing a branch command, e.g. \( \rightarrow 18 \)
STATE INDICATOR

To clear a program from a state of suspension,

enter a right-pointing arrow \( \rightarrow \)
(one for each suspension *) .

Now the state indicator is empty,

and local variables do not have values available.

Note: The state indicator is helpful in keeping track of the status of programs as you execute them.
The current definition of DRILL

```
VDRILL[18] V
V DRILL: N; W
[1] N=0
[2] NEWPROB: W+0
[3] N=N+1
[4] +(N>5)/END
[7] 4p'-'1
[8] GUESS: ANSWER+[]
[9] +(ANSWER=STOP)/0
[10] +(ANSWER=X*Y)/NEWPROB
[11] 'NO, TRY AGAIN'
[12] W=W+1
[13] +(W<3)/GUESS
[14] 'LATER. GET SOME HELP NOW'
[15] 0
[16] END: 'CONGRATULATIONS! WOULD YOU LIKE 5 MORE?'
[17] 'ENTER YES OR NO.'
[18] +(YES='Y')/1
[19] 'O.K. NO HARD FEELINGS. SEE YOU NEXT TIME.'
```

Challenge:

```
VDRILL[18]
```

DRILL may be edited to rectify the problem which caused the previous suspension.

Then execute DRILL.
A SIMPLE GAME PROGRAM

```
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&quot;LOL&quot;</td>
</tr>
<tr>
<td>2</td>
<td>'WELCOME TO THE GAME OF LAST-ONE LOSES!'</td>
</tr>
<tr>
<td>3</td>
<td>'DO YOU KNOW THE RULES?'</td>
</tr>
<tr>
<td>4</td>
<td>'L1: TO START WITH THERE ARE ')N+5?10;(' BOXES'</td>
</tr>
<tr>
<td>5</td>
<td>(2*N)o '[]'</td>
</tr>
<tr>
<td>6</td>
<td>'WANT TO GO FIRST OR SECOND?'</td>
</tr>
<tr>
<td>7</td>
<td>'L2: 'YOUR MOVE.'</td>
</tr>
<tr>
<td>8</td>
<td>'N+N-MMOVE'</td>
</tr>
<tr>
<td>9</td>
<td>'LOSExN=0'</td>
</tr>
<tr>
<td>10</td>
<td>'L3: 'MY MOVE.'</td>
</tr>
<tr>
<td>11</td>
<td>'WINxN=1'</td>
</tr>
<tr>
<td>12</td>
<td>'LOSExN=1'</td>
</tr>
<tr>
<td>13</td>
<td>'N+N-PMOVE'</td>
</tr>
<tr>
<td>14</td>
<td>'WINxN=0'</td>
</tr>
<tr>
<td>15</td>
<td>'L2: 'YOUR MOVE.'</td>
</tr>
<tr>
<td>16</td>
<td>'LOSExN=1'</td>
</tr>
<tr>
<td>17</td>
<td>'WINxN=0'</td>
</tr>
<tr>
<td>18</td>
<td>'LOSExN=1'</td>
</tr>
<tr>
<td>19</td>
<td>'WINxN=0'</td>
</tr>
<tr>
<td>20</td>
<td>'L3'</td>
</tr>
<tr>
<td>21</td>
<td>'WIN: 'I WIN THIS TIME.'</td>
</tr>
<tr>
<td>22</td>
<td>'L4'</td>
</tr>
<tr>
<td>23</td>
<td>'LOSE: 'RATFINK!!! YOU WIN.'</td>
</tr>
<tr>
<td>24</td>
<td>'L4: 'TYPE LOL TO PLAY AGAIN.'</td>
</tr>
</tbody>
</table>
```

LAST-ONE-LOSES is a variant of the ancient intellectual game, NIM.

LOL is the main program. Note the use of null literal on line [2], line labels on lines [6], [10], [15], [21], [23], and [24], a different branching format on lines [4], [9], [13], [14], etc., and the use of sub-programs.

RULES is a sub-program (self-explanatory).

PMOVE is a sub-program which accepts the player's move. Note that it checks to be sure a 1 2 or 3 is entered.

MMOVE is a sub-program which makes the computer's move. The move is simply a random number from 1 to 3, but less than or equal to N, the number of boxes. (Considerably more sophisticated strategies could be programmed here.)

INTERACTIVE PROGRAMS

114
WELCOME TO THE GAME OF LAST-ONE-LOSES!

DO YOU KNOW THE RULES?

NOPE

LAST-ONE-LOSES IS A GAME OF TAKING AWAY BOXES.
WHEN IT IS YOUR TURN, YOU MAY TAKE 1, 2 OR 3 BOXES.
YOU AND THE COMPUTER WILL TAKE TURNS TAKING BOXES AWAY
UNTIL THERE IS ONLY ONE BOX LEFT. WHOEVER TAKES THE
LAST ONE LSES!

TO START WITH THERE ARE 13 BOXES

[] [] [] [] [] [] [] [] [] [] [] []

WANT TO GO FIRST OR SECOND?

FIRST
YOUR MOVE.
[]: 2
[] [] [] [] [] [] [] [] [] [] [] []
MY MOVE.
[] [] [] [] [] [] [] [] [] [] [] []
YOUR MOVE.
[]: 3
[] [] [] [] [] [] [] [] [] [] [] []
MY MOVE.
[] [] [] [] [] [] [] [] [] [] [] []
YOUR MOVE.
[]: 2
RATFINK!!! YOU WIN.
TYPE LOL TO PLAY AGAIN.

LOL

WELCOME TO THE GAME OF LAST-ONE-LOSES!

DO YOU KNOW THE RULES?

YES
TO START WITH THERE ARE 6 BOXES
[] [] [] [] [] []

WANT TO GO FIRST OR SECOND?

SECOND
MY MOVE.
[] [] [] [] [] []
YOUR MOVE.
[]: 1
[] [] [] [] [] []
MY MOVE.
[] I WIN THIS TIME.
TYPE LOL TO PLAY AGAIN.
A SIMPLE SIMULATION PROGRAM

```plaintext
VTMPER
[1]  EMOTION=0
[2]  'HOW DO YOU FEEL ABOUT ME?'
[3]  ENTER:NEW+[]
[4]  EMOTION+NEW+EMOTION=2
[5]  +(EMOTION>10)/MAD
[6]  +ENTER
[7]  MAD: '**!?!!*!?'
```

TEMPER is a program which simulates -- albeit crudely -- an emotional reaction.

This program will -- under certain conditions -- "get mad at you"!

It begins by asking you to express how you feel toward the program. Numbers are used to indicate the strength of your feelings: low numbers are very kind or loving, high numbers are hostile or frustrating.

You may ENTER a sequence of NEW numbers -- one at a time. Each number causes EMOTION to be changed according to a simple mathematical model:

EMOTION becomes the NEW value plus one half the previous value of EMOTION.

If EMOTION ever becomes greater than 10, the program goes to MAD (where the computer's vernacular is printed)

Try different sequences (like 8 6 4 or 8 4 6 or 7 2 7 or 2 7 7, etc.)
TEMPEH

HOW DO YOU FEEL ABOUT ME?

☐:

5

☐:

5

☐:

5

☐:

5

☐:

5

☐:

5

☐:

5

☐:

5

How many 5's do you think this program can "tolerate"?
**REVIEW**

*Interactive* programs permit you to enter data during their execution. In APL, the quad □ and quote-quad □ symbols are used to request input—the latter accepting only literal input.

A drill-and-practice program is one which interacts with a student in order to improve his skills, say in multiplication. A prototype of such a program would give directions, present problems, request the student’s answers, and judge the answers for correctness. Based on whether an answer is right or wrong, the program branches and gives the appropriate response.

Line editing allows you to change single characters on a line in a program and is helpful in refining a program after it has been defined. Line labels are particularly convenient when a program is to undergo further editing changes. A line label is local to a program and takes on the value of the line number with which it is currently associated. Hence, even after new lines have been inserted or deleted, branching commands using line labels will still be valid.

When a program is executed and produces an error, it is said to be “suspended.” The remainder of the program—not yet completed—is temporarily held in abeyance. By checking the “state indicator” $SI$, you can find out where, when, and how many suspensions have occurred. Execution can be resumed with a branch command.

A game is ideal for writing as an interactive program, especially one in which a player competes against the computer. Strategies for making the computer's moves can be programmed, perhaps the simplest of which is by random selection.

A simulation is an approximation of some real-world phenomenon. Simulating something as complex as human behavior is extremely challenging, although simple mathematical models can be expressed easily in APL.
U-Program 7

ARRAYS

Contents

The Restructuring Function \( \rho \) 120
Matrices 121
The Ravel Function , 124
Functions on Arrays 125
Array Indexing 127
Summary of APL Data Structures 133
Programs Using Arrays 135
Review 138
THE RESTRUCTURING FUNCTION $\rho$

(This is the dyadic use of the rho symbol $\rho$)

$5 \rho 3$ generates an array of 5 3's

(a "vector")

5 4's

4 5's

3 5's

Generally the form is: (structure) $\rho$ (elements)

7 $\rho$ 8 9
8 9 8 9 8 9 8

6 $\rho$ 8 9 10
8 9 10 9 8 9 8 9 8

When there are too few elements, repeat them until the structure is filled up.

5 $\rho$ 8 9 10 11
8 9 10 11 8

4 $\rho$ 8 9 10 11 12
8 9 10 11

When there are too many, use only enough elements to fill the structure.

3 $\rho$ 8 9 10 11 12 13

120 ARRAYS
MATRICES

With two numbers on the left, the array produced is two-dimensional. (a "matrix")

Here, 3 rows and 4 columns of 5's

4 rows 3 columns of 8's

3 rows 5 columns of 2's

5 by 2 structure, 0s and 1s as elements fill up the matrix, row by row (restructuring a vector into a matrix)

2 rows 5 columns of the elements 1 2 3 4 ... 10
More examples of numerical matrices:

Note that the rows are filled up one at a time.

(This is called "row-major" order)
Dyadic p with literals:

5 \( p \) ' - ' 5 dashes

-----

4 \( p \) ' . ' 4 dots

....

3 \( p \) ' * ' 3 stars

\[ 20 \ p \ ' \cdot \cdot \cdot ' \]

* * * * * * * * * *

A total of 20 symbols, alternating \( \square \)s and *s.

\[ 7 \ p \ ' T O O T ' \]

A total of 7 elements, repeating when necessary

\[ 5 \ p \ ' P H O T O G R A P H Y ' \]

PHOTO

The first 5 elements

\[ 3 \ p \ ' S E X T U P L E ' \]

The first 3

\[ 12 \ p \ ' O H ! ' \]

12 elements in total

! is an overstrike symbol
Type ' backspace .
3 4 ρ 'FREEFROMDEBT'

FREE
FROM
DEBT

2 30 ρ 'AND MILES TO GO BEFORE I SLEEP'

2 by 30
(repeat 30 characters)

6 3 ρ 'TO BE OR NOT'

Repeat in order to fill up 6 by 3 matrix

L + 3 4 ρ 'GOODPLAYBILL'

L

L is specified to be a matrix of 3 rows and 4 columns

ρ L

3 4

ρ L (monadic) gives the structure of L -- 3 rows, 4 columns

THE RAVEL FUNCTION,

, L

GOODPLAYBILL

, converts a matrix (or any array) into a vector
FUNCTIONS ON ARRAYS

\[ M + 3 \quad 4 \quad \rho \quad 12 \]

\[ M \]

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

\[ \rho M \]

\[ \begin{array}{c}
3 \\
4
\end{array} \]

\[ ,M \]

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

\[ \rho , M \]

\[ \rightarrow \]

\[ M + 1 \]

\[
\begin{array}{cccc}
2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 \\
10 & 11 & 12 & 13
\end{array}
\]

\[ M \times 3 \]

\[ \rightarrow \]

\[ (,M) = (\times / \rho , M) \rho , M \]

\[ \rightarrow \]
Try these functions with MATRIX

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]

\[
\begin{bmatrix}
3 & 4 & 9 & 5 & 0 & 6 & 2 & 4 & 11 & 3 & 16 & 8 & 20 & 7
\end{bmatrix}
\]
ARRAY INDEXING

\[
\begin{array}{cccccccccccc}
9 & 5 & 0 & 6 \\
2 & 4 & 11 & 3 \\
16 & 8 & 20 & 7
\end{array}
\]

Specifying and displaying \texttt{MATRIX}

\[
\begin{align*}
\text{\texttt{MATRIX}[2;3]} & \quad \text{Indexing the } 2^{nd} \text{ row, } 3^{rd} \text{ column of } \texttt{MATRIX} \\
& \quad 11 \\
\text{\texttt{MATRIX}[1;4]} & \quad \text{The } 1^{st} \text{ row, } 4^{th} \text{ column element} \\
& \quad 6 \\
\text{\texttt{MATRIX}[3;2]} & \quad \text{The } 3^{rd} \text{ row, } 2^{nd} \text{ column element} \\
\rightarrow & \quad \\
\text{\texttt{MATRIX}[1;]} & \quad \text{The } 1^{st} \text{ row (and all columns)} \\
& \quad 9 \ 5 \ 0 \ 6 \\
\text{\texttt{MATRIX}[;3]} & \quad \text{The } 3^{rd} \text{ column (and all rows)} \\
& \quad (\text{printed as a vector}) \\
\rightarrow & \quad \\
\text{\texttt{MATRIX}[3;]} & \quad \text{The } 3^{rd} \text{ row} \\
\rightarrow & \quad \\
\text{\texttt{MATRIX}[;2]} & \quad \text{The } 2^{nd} \text{ column} \\
\rightarrow & \quad \\
\end{align*}
\]
\[ \text{The 2}^{\text{nd}} \text{ and 4}^{\text{th}} \text{ columns} \]

\[
\begin{array}{ccc}
5 & 6 \\
4 & 3 \\
8 & 7
\end{array}
\]

\[ \text{The 2}^{\text{nd}} \text{ and 3}^{\text{rd}} \text{ row elements of the 2}^{\text{nd}} \text{ and 4}^{\text{th}} \text{ columns} \]

\[
\begin{array}{ccc}
4 & 3 \\
8 & 7
\end{array}
\]

\[ \text{2}^{\text{nd}} \text{ row; 4}^{\text{th}}, 2^{\text{nd}} \text{ and 3}^{\text{rd}} \text{ columns} \]

\[
\begin{array}{ccc}
2 & 4 & 2 \\
3 & 2 & 3
\end{array}
\]

\[ \text{3}^{\text{rd}}, 2^{\text{nd}}, \text{ and 3}^{\text{rd}} \text{ rows of 3}^{\text{rd}} \text{ column} \]

\[
\begin{array}{ccc}
3 & 2 & 3
\end{array}
\]

\[ \text{Challenge:} \]

\[ \text{MATRiX}[1 \ 2 \ 3; 1 \ 2 \ 3]=\text{MATRiX}[2; 2]. \]

\[ \text{Challenge:} \]

\[ I + 0\text{MATRiX} \]

\[ \text{MATRiX}[I[1]; I[2]] \]

\[ \text{There will be 9 answers to this-- in a 3 by 3 matrix} \]

\[ \text{MATRiX}[; I[2]]=\text{MATRiX}[; I[1]]; \]
**REDUCTION WITH ARRAYS**

MAT has two dimensions (rows and columns)

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

The sum-reduction of the first dimension of MAT (adding down the columns)

+/[1] MAT

5 7 9

The sum-reduction of the second dimension of MAT (adding across the rows)

+/[2] MAT

6 15

(+/MAT)=+/[2]MAT

1 1

+/MAT is an abbreviated way of writing +/[2]MAT

(The last dimension is understood)

+/[MAT]=+/[1]MAT

1 1 1

+/MAT

+/+/MAT

21

+/.,MAT

Adding up all the elements
The product-reduction of MAT

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

\[
\timesMAT \\
4 & 10 & 18
\]

\[
\rightarrow
\]

\[
\times/MAT
\]

\[
\rightarrow
\]

\[
-\timesMAT
\]

\[
\rightarrow
\]

\[
-\times/MAT
\]

\[
2 & 5
\]

\[
(1-2-3), (4-5-6)
\]

\[
(+/+MAT) = +/-MAT
\]

\[
\rightarrow
\]

\[
(1 \times 3 \times 5 \div 2 \times 4 \times 6) = \div/MAT
\]

\[
(1 \div 2 \div 3 \div 4 \div 5 \div 6)
\]

Is this true?
THE COMPRESSION FUNCTION APPLIED TO ARRAYS

0 1 0 / 3 3 ρ 19

4 5 6

0 1 0 / 3 3 ρ 19

0 1 0 is compressed on the rows (the first dimension)

0 1 0 is compressed on the columns (the last dimension)

\[ \square + M + 4 4 \rho 'SOLDOHIOFINETOES' \]

SOLD
OHIO
FINE
TOES

\[ L + 1 1 0 0 \]
\[ K + 1 0 1 0 \]

Compress the rows

Compress the columns

ARRAYS 131
Take 🔄 and Drop ⬇ applied to matrices:

\[ \text{JIBEFORESAIL} \]

A literal matrix, MATE

\[
\begin{align*}
2 & 3 + \text{MATE} \\
\text{JIB} & \text{FOR} \\
-2 & -3 + \text{MATE} \\
\text{ORE} & \text{AIL} \\
3 & -1 + \text{MATE} \\
\text{L} & \\
2 & 3 + \text{MATE} \\
2 & 1 + \text{MATE}
\end{align*}
\]

Take the first 2 rows and the first 3 columns.

Take the last 2 rows and the last 3 columns.

Take the first 3 rows and the last 1 column.

Drop the first 2 rows and the first 3 columns.

Drop the first 2 rows and the first 1 column.
SUMMARY OF APL DATA STRUCTURES

S is a scalar (no dimension)

V is a vector (one dimension)

M is a matrix (two-dimensional)

H is a 3-array (three-dimensional)

4-arrays
5-arrays
etc.
are allowed too
\( p \) of an array is its "structure"

\[ \rho_H \]

H has 3 planes of 2 rows 4 columns

\[ \rho_M \]

M has 3 rows and 4 columns

\[ \rho_V \]

V has 6 elements (columns)

\[ \rho_S \]

S has no structure

\[ \rho\rho_H \]

H is a 3-array

\[ \rho\rho_M \]

M is a 2-array (matrix)

\[ \rho\rho_V \]

V is a 1-array (vector)

\[ \rho\rho_S \]

S is a 0-array (scalar)
PROGRAMS USING ARRAYS

\[ \text{ALPHABET} = 'abcdefghijklmnopqrstuvwxyz' \]

\[ \text{RANDOM LETTERS}\{J \}
\]

\[
\begin{align*}
&[1] \quad J \leftarrow 0 \\
&[2] \quad \text{ALPHABET}\{\text{?LETTERS}\{\text{p}\text{26}} \\
&[3] \quad J \leftarrow J + 1 \\
&[4] \quad \rightarrow (J < N) / 2V
\end{align*}
\]

\[ 7 \text{ RANDOM } 3 \]

\[
\begin{align*}
\text{QFA} \\
\text{RKG} \\
\text{LGC} \\
\text{DZT} \\
\text{KLN} \\
\text{DKD} \\
\text{NSA}
\end{align*}
\]

\[ 3 \text{ RANDOM } 7 \]

\[
\begin{align*}
\text{BNQDZUP} \\
\text{RVBMATT} \\
\text{KJCJEXL}
\end{align*}
\]

\[ \text{ALPHABET}\{\text{? 3 7 p 26}} \]

\[ \text{RANDoM} \text{ is a program which prints } N \text{ randomly generated "words" with a certain number of LETTERS.} \]

\[ \text{LETTERS p 26 generates an array of 26's which are then used as random indices of the ALPHABET.} \]

\[ J \text{ is a local name used to count up to } N. \]

\[ 7 \text{ randomly generated 3-letter "words"} \]

\[ 3 \text{ random 7-letter "words"} \]

\[ \text{A more direct way to produce the same result, using a matrix index.} \]
Program SPELLING* drills a student in spelling the numbers 1 through 8 (presented randomly).

SPELLING

SPELL 7
SEVEN
SPELL 5
FIVE
SPELL 8
ATE

THE CORRECT SPELLING IS EIGHT
SPELL 1
UNITY

THE CORRECT SPELLING IS ONE
SPELL 5
FIVE
SPELL 4
FORE

THE CORRECT SPELLING IS FOUR
SPELL 4.
FOURTEEN

THE CORRECT SPELLING IS FOUR

SPELL 8
EIGHTY

SPELL 6

W is a matrix of the correct spellings

W ← 8 5 p 'ONE TWO THREEFOUR FIVE SIX SEVEN EIGHT'

SPELLING

SPELL 7
SEVEN
SPELL 5
FIVE
SPELL 8
ATE

THE CORRECT SPELLING IS EIGHT
SPELL 1
UNITY

THE CORRECT SPELLING IS ONE
SPELL 5
FIVE
SPELL 4
FORE

THE CORRECT SPELLING IS FOUR
SPELL 4.
FOURTEEN

THE CORRECT SPELLING IS FOUR

SPELL 8
EIGHTY

SPELL 6

Hmmm.
Enter nothing and the program stops.

15 and 10 are limits for the random numbers

Here's a hint (a matrix of small circles)

You get two chances before you are told the answer.

* These programs are quite similar to ones first defined by Kenneth Nerson in his paper "The Role of Computers in Teaching", Queen's University, 1968.
REVIEW

APL treats arrays as whole entities. Any array—literal or numerical—may be restructured into another array (of the same type) in any specified size. Many APL functions, such as $+$ $-$ $\times$ $\div$ $|$ $\mid$ $=$ $\neq$ $<$ $\leq$ $\geq$ $*$ $\land$ $\lor$ $\sim$ extend to arrays; that is, the function applies to each element of the array, and the result is an array of the same size.

The term "array" includes scalars (single elements), vectors (one-dimensional arrays), matrices (two-dimensional arrays), 3-arrays, 4-arrays, etc. In APL the structure of arrays is always rectangular, and elements fill up the latter dimensions of the structure first. For matrices, this amounts to filling up row-by-row. Data structures in APL are related—as seen by the use of $\rho \rho$, the "rank" of an array.

The use of arrays-as-wholes greatly facilitates programming. Many programmers claim that APL array-handling capabilities make problem-solving considerably easier.
U-Program 8

ARRAY FUNCTIONS

Contents

The Reversal Function $\phi$ (Monadic) 140
The Transpose Function $\tau$ (Monadic) 141
The Rotation Function $\phi$ (Dyadic) 142
The Transpose Function $\tau$ (Dyadic) 145
Outer Product $\circ.f$ 148
Inner Product $f.g$ 151
Catenation of Arrays , 156
The Lamination Function , $[I]$ 158
The Matrix Inverse Function $\mathbf{B}$ (Monadic) 160
The Matrix Divide Function $\mathbf{B}$ (Dyadic) 161
Review 162
THE REVERSAL FUNCTION \( \phi \) (MONADIC)

\[
\begin{align*}
V + 'EVIL' \\
\phi V \\
\text{LIVE}
\end{align*}
\]

\( \phi 'NOSLIW' \)

\[
\phi 'DOCNOTEIDISSENTAFASTNEVERPREVENTSAPATNESSIDIETONCOD'
\]

(one of the world's longest palindromes)

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

\( \phi \) with a matrix

Each row is reversed 
(same as \( \phi[2] M \))

Each column is reversed 
(same as \( \phi[1] M \))

Reversals in both dimensions

Note: the function symbol shows the axis of reversal:

\( \phi \) (reversal about a vertical axis)

\( \theta \) (reversal about a horizontal axis)
THE TRANSPOSE FUNCTION \(\phi\) (MONADIC)

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

\[
\phi \left( M \right) = \phi M
\]

\[
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\]

Each row is transposed to a column, and each column becomes a row.

The dimensions of \(M\) are compared with the reversed dimensions of \(M\).

\(\phi N = \phi \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}\) 'FOEANDICELEN'

Form a 4 by 3 matrix \((N)\) and then print the elements transposed so that each row is a column and vice versa.

Note: the function symbol shows the axis of transposition: \(\phi\) (transpose about the main diagonal)
THE ROTATION FUNCTION $\phi$ (DYADIC)

$A \leftarrow 3$

$B \leftarrow 'TENFLAT'$

$\{ \} \leftarrow R \leftarrow A \phi B$

**FLATTEN**

$(\rho R) = \rho B$

For vectors:

$\phi$ Rotated 3 elements from the front of the elements of $B$ to the back

Compare the dimension of the result $R$ with the dimension of $B$

**Challenge:**

\[ \chi/R = (A - \rho B) \phi B \]

$1$

$-1 \phi 'TOPS'$

A clue to the use of negative numbers with rotation
Rotation works similarly with matrices:

3 columns rotated from each row (front to back)

Rotate 2 columns from the front of each row to the back

2 rows rotated from each column (bottom to top)

Rotate 1 row from the bottom to the top of each column

Challenge:

$2 \phi 1 \circ \circ 5 2 \circ \circ \text{UPCLEESAAP}$
You may rotate different numbers of elements from different rows or
columns.

Rotate 1 from the first row
2 from the second row
3 from the third row

Rotate 3 from the first column
2 from the second column
-1 from the third column
1 from the fourth column
(bottom to top)
(top to bottom)
THE TRANSPOSE FUNCTION \( \oplus \) (DYADIC)

\[ \square + L + 5 \, 8 \, p \, 'APL' \]

\[
\begin{align*}
APL & \quad APL \\
APL & \quad APL \\
APL & \quad APL \\
APL & \quad APL \\
APL & \quad APL \\
APL & \quad APL
\end{align*}
\]

\[ \square + L + 2 \, 1 \, \oplus \, L \]

\[
\begin{align*}
AAAAA \\
PPPPP \\
LLLLL
\end{align*}
\]

\[
\begin{align*}
AAAAA \\
PPPPP \\
LLLLL
\end{align*}
\]

Normal transpose
(the 2\textsuperscript{nd} dimension is switched with the 1\textsuperscript{st} dimension)

\[
(\square L) = 2 \, 1 \, \oplus \, L
\]

\[
\begin{align*}
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1
\end{align*}
\]

\[ (\rho L) = (\rho L)[2 \, 1] \]

\[
\begin{align*}
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1 \\
1 & \quad 1 \quad 1 \quad 1 \quad 1
\end{align*}
\]
A matrix $SQ$

Its transpose

$(\rho D) = L/\rho SQ$

The diagonal of this matrix

\[ \begin{bmatrix}
2 & 1 & 0 & 0 \\
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 \\
1 & 5 & 9 & 13 \\
2 & 6 & 10 & 14 \\
3 & 7 & 11 & 15 \\
4 & 8 & 12 & 16 \\
\end{bmatrix} \]
A REVIEW OF ARRAY FUNCTIONS

\[ \square \rightarrow LM \rightarrow 2 \rightarrow 2 \rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0 \]

\[ \phi_L M \]

\[ \phi \phi_L M \]

\[ \phi \phi \phi_L M \]

\[ 0 \rightarrow 0 \rightarrow \phi \rightarrow LM \]

\[ 0 \rightarrow 1 \rightarrow \theta \rightarrow LM \]

\[ 1 \rightarrow 1 \rightarrow \phi \rightarrow LM \]
OUTER PRODUCT

Outer product is used to create arrays by performing a dyadic function on every pair of elements given on the left and right.

Normal element-by-element addition is not possible because vectors $X$ and $Y$ are of different lengths.

Outer product gives all the sums of each element of $X$ with each element of $Y$ -- arranged in a table (matrix).

Other dyadic functions may be used with outer product.

Fill in this table:

$$
\begin{array}{c|ccccc}
\text{X} & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
2 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
3 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
4 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
\end{array}
$$

$$
\begin{array}{c|ccccc}
\text{Y} & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
2 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
3 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
4 & \text{=0} & \text{=0} & \text{=0} & \text{=0} & \text{=0} \\
\end{array}
$$

$$
\begin{array}{c|ccccc}
\text{array product of } X \times Y \\
\hline
1 & (1=1) & (1=2) & (1=3) & (1=4) & (1=5) \\
2 & (2=1) & (2=2) & (2=3) & (2=4) & (2=5) \\
3 & (3=1) & (3=2) & (3=3) & (3=4) & (3=5) \\
4 & (4=1) & (4=2) & (4=3) & (4=4) & (4=5) \\
\end{array}
$$

148 ARRAY FUNCTIONS
The general form of outer product is \((\text{array}) \odot f(\text{array})\)

where \(f\) is any dyadic function which extends to arrays.

\[ (+-x\div*=<\leq\geq\neq\&\mid\perp\text{etc.}) \]

Note that the size of the result is \((\rho x), \rho y\)

$$X \odot.* Y$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>27</td>
<td>81</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>81</td>
<td>256</td>
<td>1024</td>
<td></td>
</tr>
</tbody>
</table>

$$\rho Y \odot.* X$$

$$X \odot.\| Y$$

$$Y \odot.< Y$$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

ARRAY FUNCTIONS 149
The multiplication table

"Printing Precision"
A system variable to change the number of significant digits in output to 4. (Previously it was 10)

The integers from 1 to 10, their squares and square roots in a table.

Changing the printing precision back to 10

Challenge:

\[
\text{VEGETABLE} + \text{PEARS} \cdot \cdot \cdot \text{'APPLES'}
\]

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

\[
(\sqrt{\text{VEGETABLE}}) / \text{PEARS}
\]
INNER PRODUCT \( f \cdot g \)

Inner product is an operation which reduces arrays by applying two dyadic functions.

The notation is:

\((\text{array}) \ f \cdot g \ (\text{array})\)

where \( f \) and \( g \) are dyadic functions which extend to arrays.

For vectors, inner product is the same as the first function reduced over the result of the second function.

For a vector and a matrix, inner product is the first \( f \) reduced over the result of the second \( g \) applied to the vector and each column of the matrix.

\[
\begin{array}{c|ccc}
+ \cdot \times & 3 & 6 & 4 \\
\hline
10 & 5 & 60 & 84 \\
\end{array}
\]

\[
(+/ \times \ M[;1]), +/V \times M[;2]
\]

Other dyadic functions used in inner product:

Minimum sums

Sums of maximums

ARRAY FUNCTIONS 151
For a matrix and a vector, inner product is the first function reduced over the results of each row of the matrix applied with the second function to the vector.
For two matrices, inner product is the first function reduced over the results of each row of the left matrix applied with the second function to each column of the right matrix.

(This is the conventional "matrix product" in linear algebra)

A 3 by 4 matrix inner product with a 4 by 5 matrix

The result is a 3 by 5 matrix

\[
\begin{pmatrix}
M & N \\
3 & 4 \\
4 & 5
\end{pmatrix}
\]

- The two matrices must be "conformable" here

The first element of the result (1st row, 1st column)

Another element of the result (second row, third column)

\[
R[2;3] = +/M[2;] \times N[;3]
\]

\[
R[;5] = (+/M[1;] \times N[;5]), (+/M[2;] \times N[;5]), +/M[3;] \times N[;5]
\]

1 1 1

The fifth column of the result

ARRAY FUNCTIONS
Try this inner product

What is its size?

You can get Q by filling in the rest of this table.
(Q[2;3] is already done.)
\[ \text{Inner product demonstrated with 3-arrays} \]

The size of the result

4-dimensional
Catenation of Arrays,

\[ \mathbf{M} \rightarrow \mathbf{M} + \begin{bmatrix} 3 & 4 & 0 \end{bmatrix} \]

\( M, 0 \)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
0 & 0 & 0 & 0
\end{array}
\]

Catenating a single element to a matrix:
(Along the last dimension, the element extends into a new column)

Catenate 0 onto the front of \( M \)

\[ \mathbf{M} \rightarrow \begin{bmatrix} 0 & M \end{bmatrix} \]

\( M, 0 \)

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
0 & 0 & 0 & 0
\end{array}
\]

Catenating a single element to a matrix:
(Along the first dimension, the elements extends into a new row)

Catenate 0 onto the top of \( M \)
$M, \begin{bmatrix} -1 & -2 & -3 \\
\end{bmatrix}$

Where are these three elements catenated?

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

These four elements are catenated as a new row (the 1st dimension)

$M, M$

Matrix $M$ catenated to matrix $M$
(Note that they are conformable)

The resulting new size

Catenate $M$ to $M$ along the 1st dimension

New rows.
THE LAMINATION FUNCTION, \([I]\)

The same symbol as for catenation is used, but with a fractional subscript.

Lamination creates a new dimension in the result by "stacking" arrays.

(The arrays must be conformable)

\([.5]\) means create a new dimension before the 1st dimension

\([2.5]\) means create a new dimension after the 2nd dimension

The size of the new array

Laminating two vectors horizontally

Laminating vertically
A REVIEW OF Catenation and Lamination

\[ \square + L + 4 \rightarrow 'ABCDEFHIJLK' \]

\[ L, '\ast' \rightarrow \]

\[ '\ast' \rightarrow L \]

\[ '\ast' \rightarrow ('\ast', L, '\ast') \rightarrow '\ast' \]

*****
*ABC*
*DEF*
*GHI*
*JKL*
*****

\[ L, [1] \rightarrow \]

\[ L, [2] \rightarrow \]

\[ L, [.5] \rightarrow \]

\[ L, [1.5] \rightarrow \]

\[ L, [2.5] \rightarrow \]

ARRAY FUNCTIONS
THE MATRIX INVERSE FUNCTION \(\mathbb{I}(\text{MONADIC})\)

\[ C + 3 \quad 3 \quad p \quad 2 \quad -1 \quad 5 \quad 1 \quad 2 \quad 1 \quad 4 \quad 0 \quad -1 \]

\[ \mathbb{I} \]

\[ C \]

\[
\begin{bmatrix}
0.04081632653 & 0.02040816327 & -0.2244897959 \\
-0.1020408163 & 0.4489795918 & -0.0612244898 \\
0.1632653061 & 0.08163265306 & -0.1020408163
\end{bmatrix}
\]

\[ C \times \mathbb{I} \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[ (\mathbb{I} \times C) \times C \]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

The \(\mathbb{I}\) symbol is formed by overstriking \(\div\) and \(\square\).

\(\mathbb{I}\) produces the "inverse" of \(C\).

The inner product of \(C\) and the inverse of \(C\) is an identity matrix.
THE MATRIX DIVIDE FUNCTION \( \otimes \) (DYADIC)

SOLVING SIMULTANEOUS LINEAR EQUATIONS

\[
\begin{bmatrix}
0 & 3 & 3 & 0 & 2 & -1 & 5 & 1 & 2 & 1 & 4 & 0 & -1
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & + & B & = & 13 & 0 & 11
\end{bmatrix}
\]

\[
(C \otimes C) + .x \ B
\]

\[
3 \ 2 \ 1
\]

\[
B \otimes C
\]

\[
3 \ 2 \ 1
\]

If a matrix \( C \) represents the coefficients of a set of linear equations:

\[
2x_1 - x_2 + 5x_3 = 13
\]

\[
x_1 + 2x_2 + x_3 = 0
\]

\[
4x_1 - x_3 = 11
\]

and a vector \( B \) represents the constants,

the solution set is found by the matrix (inner) product of the inverse of the coefficients matrix and the constants vector.

Dyadic use of \( \otimes \) (the "matrix division" function) yields the solutions directly.

\[
x_1 = 3
\]

\[
x_2 = -2
\]

\[
x_3 = 1
\]
REVIEW

Several primitive APL functions are especially designed for array operations. For instance, the reversal \( \Phi \) (monadic) and rotation \( \Phi \) (dyadic) functions rotate elements of an array. The transpose \( \Theta \) (monadic and dyadic) function interchanges specified dimensions of an array and provides a convenient expression for array diagonals.

Two special operators—"outer product" and "inner product"—are particularly powerful. Outer product provides an alternate way to create arrays by performing a specified dyadic function on every pair of elements drawn from the arrays on the left and right. Inner product reduces two arrays by applying two dyadic functions—the first function reduced over the result of the second. \( + \times \) is the conventional "matrix product." Both outer product and inner product are generalized to accept any dyadic functions which extend to arrays.

The catenation function—previously used with vectors and scalars—also applies to arrays. New rows or columns may be appended, depending on the dimension indicated in brackets \([\ ]\). The \( , \) symbol may also be used to laminate arrays, stacking them along a specified dimension.

Finally, an overstrike symbol \( \mathbf{I} \) is the matrix inverse/matrix divide function. It can only be used with matrices—to find their inverses (monadically) or to solve linear equation systems and least squares regressions (dyadically).
U-Program 9

MISCELLANEOUS

Contents

Arithmetic Functions + - × ÷ * (Monadic)  164
Logarithmic Functions e  168
Circular Functions a  170
Recursion  176
Factorial Function! (Monadic)  178
Combinations Function! (Dyadic)  178
Base Value Function @  179
Representation Function T  182
Execute Function $  186
Format Function %  186
Scan Operator \  188
Expansion Function \  189
Review  191
ARITHMETIC FUNCTIONS $+ - \times \div \star$ (MONADIC)

$\square + V + + 2 - 2 \ 3.7 \ 0 \ -12 \ 13$
$2 - 2 \ 3.7 \ 0 \ -12 \ 13$

$\square + V + + 2 - 2 \ 3.7 \ 0 \ -12 \ 13$
$2 - 2 \ 3.7 \ 0 \ 12 \ -13$

Identity Function $+$

The result is identical to the numbers on the right

$\square + w + - 2 - 2 \ 3.7 \ 0 \ -12 \ 13$
$2 - 2 \ 3.7 \ 0 \ 12 \ -13$

Negation Function $-$

The result is the negation of the numbers on the right

same as $0 + 5$

same as $0 - 6$
The result indicates the sign of the number(s) on the right:

1 for a positive number
0 for zero
-1 for a negative number

This expresses the signum function for any number(s) B
RECIPROCAL FUNCTION $\div$

The result is the reciprocal of the number(s) on the right.

\[ \div 2 \]
0.5

\[ \rightarrow \]
\[ \div 3 \]
same as $1 \div 3$

\[ \div 4 \]
5 0.125 7

0.25 0.2 8 0.1428571429

\[ \rightarrow \]
\[ \div 10 \]

\[ \rightarrow \]
\[ \div 100 \]

0.01
**EXOENENTIAL FUNCTION**

\[ e^{1} = 2.718281828 \]

\[ e^{2} = 7.389056099 \]

\[ 2.718281828 \times 2 \quad 7.389056096 \]

\[ 2.718281828 \times 1 \]

\[ PO WERS + 18 \]

\[ N + 10 \times PO WERS - 1 \]

\[ (1 + \frac{1}{N}) \times N \]

Using 8 powers of 10 for numbers \( N \),

this expression converges on the value of \( e \)

\[ 2 \quad 2.59374246 \quad 2.704813829 \quad 2.712923932 \quad 2.718145927 \]

\[ 2.71828237 \quad 2.718280469 \quad 2.718281688 \]
LOGARITHMIC FUNCTIONS

\[ \begin{align*}
\bullet & \ 7.389056099 \ 2.718281828 \\
2 & 0.9999999998 \\
*2 & 1 \\
7.389056099 & 2.718281828 \\
\bullet & \ 3 \\
\rightarrow & \\
3 & \\
B + 10 & \\
N + 100 & \\
\bullet N & \\
4.605170186 & \\
\bullet B & \\
2.302585093 & \\
(\bullet N) \div \bullet B & \\
\rightarrow & \\
(B \bullet N) = (\bullet N) \div \bullet B & \\
1 & \\
B \bullet N & \\
\rightarrow & \\
\text{The dyadic use of } \circ \text{ is equivalent to dividing the natural logarithms of the Number } N \text{ and the base } B.
\end{align*} \]

overstrike 0 and *

The Natural Logarithm (monadic \(\circ\))

The results are the natural logarithms of the numbers on the right.

2 and (1) are the powers to which you must raise e to get the numbers shown.

What is the natural logarithm of e raised to the 3 power?

e raised to the \(3\) power is 3

A base B and a number N
THE LOGARITHMIC FUNCTION (dyadic ⋆)

The result is the power to which the number on the left must be raised in order to get the number on the right.

(The number on the left is usually called the “base”)

What is the logarithm of 10*2 to the base 10?
and the logarithm of 1000 to the base 10?

What powers of 10 are these?

The logarithm of 25 to the base 5 is 2. (5 raised to the 2 power is 25)

What is the logarithm of 7*3 to the base 7?

What is the logarithm of 81 to the base 3?

The logarithm of 1 to any positive base is 0
CIRCULAR FUNCTIONS

The Pi-times function (monadic 0)

The result is pi (π) times the number on the right.

\[ \pi \times 2 \]

...and the number displayed in the output.

Now only 3 digits are printed

\[ \pi \times \text{each of the RADII squared} \]
The Trigonometric Functions (dyadic 0)

The result is evaluated for the particular trigonometric function indicated on the left where 1 indicates sine, \( \frac{2}{3} \) cosine, \( \frac{3}{3} \) tangent etc.
and the number(s) on the right are expressed in radians.

The sine of \( \pi \) radians (180°) is close to 0.

The sine of \( \frac{\pi}{4} \) radians (45°)
(to 3 significant digits)

\[ \theta \rightarrow \square + \text{RADIANS} + 0 \div 6 \div 12 \]

several angles: \( \frac{\pi}{6}, \frac{\pi}{3}, \frac{\pi}{2}, \frac{2\pi}{3}, \frac{5\pi}{6}, \pi, \frac{7\pi}{6}, \frac{4\pi}{3}, \frac{3\pi}{2}, \frac{5\pi}{3}, \frac{11\pi}{6}, 2\pi \)

The sine of each angle

\[ \text{ANGLE} \rightarrow \text{RADIANS} \]

0.5 0.866 1 0.866 0.5 0 -0.5 -0.866 -1 -0.866 -0.5 0

The cosine of each angle

\[ \text{ANGLE} \rightarrow \text{RADIANS} \]

0.866 0.5 0 -0.5 -0.866 -1 -0.866 -0.5 0 0.5 0.866 1

The tangent of each angle

For any one of the angles,

\[ \text{ANGLE} + \text{RADIANS}[? \div \text{RADIANS}] \]

The tangent of the ANGLE equals the sine of the ANGLE divided by the cosine of the ANGLE.

\[ (3 \circ \text{ANGLE}) = (1 \circ \text{ANGLE}) \div 2 \circ \text{ANGLE} \]
It may be convenient to embody these trigonometric functions as monadic defined functions.

To make this family of functions complete, the following are available.

\[
\begin{align*}
\sin Z &= 1 + \text{ANGLE} \\
\cos Z &= 2 + \text{ANGLE} \\
\tan Z &= 3 + \text{ANGLE} \\
\sinh Z &= 4 + \text{ANGLE} \\
\cosh Z &= 5 + \text{ANGLE} \\
\tanh Z &= 6 + \text{ANGLE} \\
\end{align*}
\]

\[
\begin{align*}
(\sin Z) &= (1 + \text{ANGLE} \times 2) \times 0.5 \\
(\cos Z) &= (1 - \text{ANGLE} \times 2) \times 0.5 \\
(\tan Z) &= (1 + \text{ANGLE} \times 2) \times 0.5 \\
(\sinh Z) &= (1 - \text{ANGLE} \times 2) \times 0.5 \\
(\cosh Z) &= (1 + \text{ANGLE} \times 2) \times 0.5 \\
(\tanh Z) &= (1 - \text{ANGLE} \times 2) \times 0.5 \\
\end{align*}
\]

The hyperbolic trigonometric functions
\begin{align*}
\text{VANGLE} &+ \text{ARCSINE} \; X \\
\text{ANGLE} &+ -1 \circ X \\
\text{VANGLE} &+ \text{ARCCOSINE} \; X \\
\text{ANGLE} &+ -2 \circ X \\
\text{VANGLE} &+ \text{ARCTANGENT} \; X \\
\text{ANGLE} &+ -3 \circ X \\
\text{VANGLE} &+ \text{ARCTRIG} \; X \\
\text{ANGLE} &+ -4 \circ X \\
\text{VANGLE} &+ \text{ARCSINH} \; X \\
\text{ANGLE} &+ -5 \circ X \\
\text{VANGLE} &+ \text{ARCCOSH} \; X \\
\text{ANGLE} &+ -6 \circ X \\
\text{VANGLE} &+ \text{ARCTANH} \; X \\
\text{ANGLE} &+ -7 \circ X \\
\end{align*}

The arc-trigonometric functions

where $-40X$ is
\((-1 + X \times 2) \times .5\)
Defined programs for converting angles in degrees to radians and vice versa.

\[
\text{DEGREES } 45 \\
0.785
\]

\[
\text{SINE DEGREES } 30 \\
0.5
\]

\[
\text{DEGREES } 60 \\
\rightarrow
\]

\[
\text{COSINE DEGREES } 60 \\
1
\]

\[
(2 \times 60 \times 60 \div 180) = \text{COSINE DEGREES } 60
\]

\[
(\text{DEGREES } 45) \times 360 \div 2
\]

\[
\frac{90}{2} = 45
\]
This attempt at defining a program produces a DEFINITION ERROR because the name RADIANS has already been assigned! (see p. 171)

DEGREES + RADIANS ANGLE
DEFN ERROR

DEGREES+RADIANS ANGLE

RADIANS

0.524 1.05 1.57 2.09 2.62 3.14 3.67 4.19 4.71 5.24 5.76 6.28

)ERASE RADIANS

DEGREES + RADIANS ANGLE
DEGREES + ANGLE:0÷180

RADIANS 01
180

RADIANS 0÷4

(0÷3) = DEGREES RADIANS 0÷3
1

30 = RADIANS DEGREES 30

The )ERASE system command will erase any names—including programs—listed immediately afterward.

RADIANS is erased, so a new program with that name can now be defined.
Pi radians is equivalent to 180 degrees

\[ \frac{\pi}{4} \text{ radians} = \ ? \text{ degrees} \]

An identity
RECURSION

Recursion is a process which in its description refers to itself -- hence causing repeated use of the same process. For example:

The definition of MEMBER includes itself in its own definition. MEMBER is a recursive program which determines whether the element(s) on the left are members of those on the right. (same as dyadic e)

\[ \forall z \in \text{MEMBER}(b) \]

\[ \rightarrow (0 = \rho b)/z \rightarrow 0 \]

\[ z + (A = 1 + B) \lor \text{MEMBER}(1 + b) \]

\[ 4 \text{ MEMBER } 15 \]

\[ T \text{ MEMBER } + 12 \]

\[ y + 12 + x + 3 \]

\[ x \text{ MEMBER } y \]

The first execution of MEMBER
the second
the third
the fourth
the fifth
the sixth terminates immediately on line [1]
the fifth terminates (with an explicit result of 0)
the fourth-terminates ("""
the third terminates (with an explicit result of 1)
the second terminates ("""
the first terminates ("""

the final result -- yes, \( x(3) \) is a member of \( y(1 2 3 4 5) \)

Untracing

Execute MEMBER for the same \( x \) but twice the \( y \).

'MEMBER 'WORD'

MEMBER with literals

'RAW' MEMBER 'WORD'

1 0 1

176 MISCELLANEOUS
FAC is also recursive. It is a program which computes the "factorial" of nonnegative integer N by repeated multiplication.

Z (the final resultant) is the value of N times the result of the execution of FAC for N-1

\[ \text{FAC } 5 \]
\[ 120 \]

\[ \text{FAC } 4 \]

\[ \&(\text{FAC } 3)=(3\times\text{FAC } 2),(3\times2\times\text{FAC } 1),(3\times2\times1\times\text{FAC } 0) \]
\[ 1 \]

\[ \text{FAC } 0 \]

\[ \times/13 \]

\[ \times/14 \]

\[ (5) = \times/15 \]
\[ 1 \]

\[ \text{!}^5 \]
\[ 5 \times 4 \times 3 \times 2 \times 1 \]
FACTORIAL FUNCTION! (MONADIC)

The result is the product of the integers from 1 to the number given on the right.

Factorials of 5, 3, and 1

Factorials of 4 and 2?

Factorial of 12 is (approx.) $4.79 \times 10^8$

Estimate this one.
(This function extends to fractions)

COMBINATIONS FUNCTION! (DYADIC)

The result is the number of combinations which may be formed by taking the number of things on the left from 2 the number of things on the right.

How many combinations can be formed from 8 things, taking 2 at a time?

$3 \, ! \, 8 = \left( \frac{8}{3} \right) \times \frac{8-3}{1} = \frac{8}{3} \times 5$

Equivalent expressions for the combinations of 8 things, taking 3 at a time.

What value does this have?

What meaning could this have?
BASE VALUE FUNCTION

The result is the value of the number on the right expressed in the base of the number on the left.

Numbers in base 2 are converted into their decimal values.

\[(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + 1 \times 2^0\]

\[(1 \times 5^3) + (4 \times 5^2) + (2 \times 5^1) + 3 \times 5^0\]

The base 5 value of 1423
1339 = +/2 4 7 3 × 8 * 3 2 1 0

Base 8

8 ⊥ 2 4 7 3

1339

(8 ⊥ 2 4 7 3) - 8 ⊥ 2 4 6 3

→

(8 ⊥ 2 4 7 3) - 8 ⊥ 2 3 7 3

→

(8 ⊥ 2 4 7 3) - 8 ⊥ 1 1 1 1

→

10 ⊥ 1 7 7 6

Base 10

A scalar on the left extends to a vector on the right, or matches (in size) a vector on the left.
The base value function can be used with several different bases. For example, 24 (hours/day) × 60 (min./hour) × 60 (sec./min.) = the number of seconds in 1 hour 3 minutes 2 sec.

Notice that the left-most element of A does not affect the result.

Another way of looking at it -- W is a "weight" vector.

This example shows (more generally) how W may be produced.

The number of seconds in 2 days 10 hours 3 min. 4 sec. (208984) -- found by base value -- is identical to the inner product of W and B.

The number of inches in 5 yards 2 feet 6 inches (Note that 12 is inches per foot, 3 is feet per yard, and the 1780 is inconsequential).

Use T to convert back to yards, feet and inches.

Convert 3782 seconds into __ hours __ minutes __ sec.
REPRESENTATION FUNCTION $\tau$

The result is the representation of the number on the right in the base system on the left. Base-10 representation of 360

What is the base-10 representation of the base-10 value of 2001 ?

The representation is truncated if there are not enough places made available on the left of $\tau$.

Numbers in base 10
represented in base 2

$360$

$(4 \ p \ 10) \ \tau \ (4 \ p \ 10) \ \downarrow \ 2 \ 0 \ 0 \ 1$

$(5 \ p \ 2) \ \tau \ 23$

$(3 \ p \ 2) \ \tau \ 23$

$2 \ 2 \ 2 \ \tau \ 5$

$2 \ 2 \ 2 \ \tau \ 4$

$2 \ 2 \ \tau \ 3$

$2 \ 2 \ \tau \ 2$

$2 \ \tau \ 1$

$2 \ \tau \ 0$
With a vector on the right, each element is represented in the bases on the left (and displayed as columns.)

What are the base 2 representations of 4 5 6 7 and 8?

Notice that there are not enough places to represent 8 fully, so it is truncated.

For N of 3,
evaluate this expression

Its transpose reveals a familiar pattern -- base 2 representations of successive integers -- found in the rows.
TABLE = TRUTH N

\[ \text{TABLE} = \Phi (N+2) \cap (12N-1) \]  

1. Display the TRUTH table of order 2

2. The TRUTH table of order 3

3. What is the size of the TRUTH table of order 4?

TRUTH is a program which produces a table of all the logical (base 2) combinations of order N.
\[ \text{DECODE is a recursive definition of the base-value function (1).} \]

[1] \( \text{VALUE} \leftarrow 1 \text{VECTOR} \)

[2] \( \rightarrow (0=\text{BASES}+1+\text{BASES})/0 \)

[3] \( \text{VALUE} \leftarrow (\text{VALUE} \times \text{BASES}) + \text{BASES} \text{ DECODE 1+VECTOR} \)

\[ \text{Note: } \n \text{DECODE} .5 \text{BASES} \rightarrow (\text{VECTOR})^2 \text{BASES} \]

will permit scalar BASES.

1780 3 12 DECODE 5 2 6

How many inches in 5 yards, 2 feet, 6 inches?

\[ \text{ENCODE is a recursive definition of the representation function (1).} \]

[1] \( \text{VECTOR} + \text{BASES} | \text{VALUE} \)

[2] \( \rightarrow (0=\text{BASES}+1+\text{BASES})/0 \)

[3] \( \text{VECTOR} + (\text{VALUE} \times \text{BASES}), \text{BASES} \text{ ENCODE } (\times / \text{BASES}) | \text{VALUE} \)

\[ \text{Note: scalar VALUE only} \]

1780 3 12 ENCODE 210

How many yards, feet, inches in 210 inches?
EXECUTE FUNCTION ¥

Normal execution of a numeric expression

" " a literal "

Converting a literal to a numeric

¥ strips off the quote marks
and executes the expression inside

¥ can be used to assign variables
or execute programs under
certain conditions

FORMA T FUNCTION ¥

Converting a numeric to a literal

(spaces included)

¥ has the effect of putting
quote marks around the value of the
expression and representing it in
the simplest literal format.

¥ and ¥ are (kind of) inverses
**DYADIC FORMAT**

ψ has a dyadic usage:

- to represent values (as literals) in a specified format.
- 1 place after the decimal point

Display 2 places after the decimal point

---

### General Formatting

\( \psi \) can be used with a vector left argument:

\[ \psi \left[ 1 \right] \text{ is the number of spaces in the horizontal field for each element of } A \]

\[ \psi \left[ 2 \right] \text{ indicates how many places after the decimal point (\(+\)) or use of E-notation (\(-\)).} \]

\[ \psi.v.A \]

The integers from 1 to 10, their squares & square roots formatted in a table with 10 columns each and 2 places after the decimal point for each number.

[Compare with table on p. 150]
SCAN OPERATOR \ 

+/10 

Sum reduction

Sum scan

gives all the cumulative sums.

(+/1), (+/2), (+/3), (+/4), (+/5), (+/6), (+/7), (+/8), (+/9), +/10

→

(×/10) = !10

1 1 1 1 1 1 1 1

|/10

\ may be used

with any dyadic scalar function

→

\0 0 1 0 1 1 0

→

\1 1 0 1 0 0 1

An alternative definition

for DECODE (⊥)

using scan \ 

(This program generalizes to ARRAYS.)
EXPANSION FUNCTION

1 0 1 0 1 / 'ABCDE'

ACE

1 0 1 0 1 / 'ACE'

A C E

\[ a + R + (Q + 1 0 1 0 1 1 0 0) / V + 2 3 5 7 11 13 17 19 \]

2 5 11 13

Q \ R

2 0 5 0 11 13 0 0

\[ \rho R \]

→

\[ +/Q \]

→

\[ \rho Q = \rho Q\ R \]

→

L + 1 1 1 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1

L \ 'BACKSLASHEXPAND'

→

The result is an array expanded to the size of the numbers on the left (always 0s and 1s) with spaces (for literals) or 0s (for numericals) inserted in the array on the right.

Expansion \ and compression \ are related.

Spaces are inserted in the literal result.

A numerical vector is compressed.

A numerical vector is expanded: elements of R corresponding to 1s in Q are preserved, 0s replace the others.

What is the size of R?

How many 1s in Q?

The size of Q and Q \ R compared.

Preserve the elements of the right where there are 1s in L; spaces replace the elements where there are 0s.
Expansion with matrices

Expanding the last dimension (columns)

Expanding the first dimension (rows)

(12 33 100) \"APL\" ; (3 10) \# 212 \times 30

An expression of tribute to a certain computer system...
Numerous functions are available in APL for special purposes: monadic use of + - × ÷, the logarithm \( \circ \) function (monadic and dyadic), pi-times \( \pi \) (monadic), all the circular functions \( \circ \) (dyadic)—including sine, cosine, and tangent—factorial and combinations \( ! \), base value \( \perp \) and representation \( \tau \), execute \( \& \), format \( \$ \), scan \( \backslash \), and expansion \( \backslash \).

Recursive programs may be defined in APL by including a program name within its own program definition. Such a program will execute itself in the process of executing itself, etc. Recursion is an extremely powerful programming technique . . . and a powerful concept!
APL Bogglers

Contents

A collection of APL expressions which boggle the mind. Some are special cases; some are implementation anomalies; and some are open mathematical questions—but all are syntactically allowable APL expressions which have results you can try to predict.
0 ÷ 0

0 × 0

1 ⊗ 1

0 ! 0

0 + 10

(10) = 10

(10) p 1

1 ⊢ 10

6 ⊢ 15

6 ⊢ 15
<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>+/10</td>
</tr>
<tr>
<td>Subtract</td>
<td>-/10</td>
</tr>
<tr>
<td>Multiply</td>
<td>×/10</td>
</tr>
<tr>
<td>Divide</td>
<td>÷/10</td>
</tr>
<tr>
<td>Equal</td>
<td>=/10</td>
</tr>
<tr>
<td>Greater</td>
<td>&gt;=/10</td>
</tr>
<tr>
<td>Less</td>
<td>&lt;=/10</td>
</tr>
<tr>
<td>More</td>
<td>&gt;/10</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;/10</td>
</tr>
<tr>
<td>Not equal</td>
<td>≠/10</td>
</tr>
<tr>
<td>Less than or equal</td>
<td>&lt;=/10</td>
</tr>
<tr>
<td>Greater than or equal</td>
<td>&gt;=/10</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;/10</td>
</tr>
<tr>
<td>Less than or equal</td>
<td>&lt;/10</td>
</tr>
<tr>
<td>Not equal</td>
<td>≠/10</td>
</tr>
</tbody>
</table>
A ← 10p5
A[10p1] ← 110
A

V ← 2 3 5 7
V[ ]
V[10]
V[ ]+9
V
V[10]+9
V
V[3]+10
V

M ← 3 4 p 112
ρM[1;]
ρM[;2]
ρM[1 3;2 4]←1+M[1 3;2 4]×9
1 0 + M
1 0 + M
0 0 + M
0 0 + M
4 5 + M
4 5 + M
2 2 + M
\[ S + 5 \]
\[ (S+8) \times S \]

\[ T + 8.8 \]
\[ (T+5) \times \lceil T \rceil \]

\[ \forall Z \leftarrow F \ X \]
\[ \left[ 1 \right] \ Z \leftarrow X \times Y \]
\[ \forall \]

\[ \forall Z \leftarrow G \ Y \]
\[ \left[ 1 \right] \ Z \leftarrow F \ Y \]
\[ \forall \]

\[ Y \leftarrow 3 \]
\[ G \ 4 \]
# Summary of APL*

## Contents

APL Primitive Functions and Commands

<table>
<thead>
<tr>
<th>Summary of Dyadic Scalar Functions</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary of Monadic Scalar Functions</td>
<td>202</td>
</tr>
<tr>
<td>Summary of Mixed Functions</td>
<td>203</td>
</tr>
<tr>
<td>Summary of Operators</td>
<td>206</td>
</tr>
<tr>
<td>Summary of Commands</td>
<td>207</td>
</tr>
<tr>
<td>Summary of Program Definition Syntax</td>
<td>208</td>
</tr>
<tr>
<td>Summary of Fundamental Programming Concepts</td>
<td>209</td>
</tr>
</tbody>
</table>

System Facilities

| Summary of Error Reports* | 211 |
| Summary of Editing Procedures* | 212 |
| List of System Commands* | 213 |
| List of System Variables* | 214 |
| List of System Functions* | 215 |

*Note: These summaries cover the material presented in this book but are incomplete in some places (marked *). For complete details consult a reference manual such as APL/360 Reference Manual, 2nd edit., by Sandra Pakin, S.R.A., 1972; or APLUM Reference Manual, 2nd edit., by Clark Wiedmann, University of Massachusetts, 1977.
"Dyadic scalar functions" are those dyadic APL primitive functions that extend the way they perform on scalars to higher order arrays.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Plus</td>
<td>Standard</td>
<td>(3+4)=7</td>
</tr>
<tr>
<td>-</td>
<td>Minus</td>
<td>Arithmetic</td>
<td>(8-3)=5</td>
</tr>
<tr>
<td>×</td>
<td>Times</td>
<td></td>
<td>(2×3)=6</td>
</tr>
<tr>
<td>÷</td>
<td>Divide</td>
<td></td>
<td>(10÷5)=2</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
<td></td>
<td>(4=4)=1</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less-Than</td>
<td>Result is 1 if the relation is true; result is 0 if the relation is false.</td>
<td>(4&lt;5)=1, (4≤3)=0, (4≥3)=1, (4&gt;5)=0, (4≠4)=0</td>
</tr>
<tr>
<td>≤</td>
<td>Not Greater</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≥</td>
<td>Not Less</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater-Than</td>
<td></td>
<td></td>
</tr>
<tr>
<td>≠</td>
<td>Not Equal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⌈</td>
<td>Maximum</td>
<td>Result is the larger</td>
<td>(5⌈8)=8</td>
</tr>
<tr>
<td>⌊</td>
<td>Minimum</td>
<td>Result is the smaller</td>
<td>(5⌊8)=5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Residue</td>
<td>Result is remainder [(A</td>
</tr>
<tr>
<td>∧</td>
<td>And</td>
<td></td>
<td></td>
</tr>
<tr>
<td>∨</td>
<td>Or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⊻</td>
<td>Nand</td>
<td></td>
<td></td>
</tr>
<tr>
<td>⊻</td>
<td>Nor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Power</td>
<td>Exponentiation</td>
<td>(3*2)=9</td>
</tr>
<tr>
<td>®</td>
<td>Logarithm</td>
<td>(Base) log (Number)</td>
<td>(3®9)=2</td>
</tr>
<tr>
<td>!</td>
<td>Combinations</td>
<td></td>
<td>(3!8)=56</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>NAME</td>
<td>DEFINITION</td>
<td>EXAMPLE</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
<td>------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0</td>
<td>Circular</td>
<td>$10(\text{radians})$ is Sine</td>
<td>$(1001) = 0$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$20(\text{radians})$ is Cosine</td>
<td>$(2001) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$30(\text{radians})$ is Tangent</td>
<td>$(300;4) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$40(\text{radians})$ is $(1+(\text{radians})^2)^{.5}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$50(\text{radians})$ is Hyperbolic Sine</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$60(\text{radians})$ is Hyperbolic Cosine</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$70(\text{radians})$ is Hyperbolic Tangent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-70(\text{number})$ is Arc Hyperbolic Tangent</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-60(\text{number})$ is Arc Hyperbolic Cosine</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-50(\text{number})$ is Arc Hyperbolic Sine</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-40(\text{number})$ is $(-1+(\text{number})^2)^{.5}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-30(\text{number})$ is Arc Tangent</td>
<td>$(-301) = 0;4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-20(\text{number})$ is Arc Cosine</td>
<td>$(-201) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$-10(\text{number})$ is Arc Sine</td>
<td>$(-100) = 1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$00(\text{number})$ is $(1-(\text{number})^2)^{.5}$</td>
<td></td>
</tr>
</tbody>
</table>
"Monadic scalar functions" are those monadic APL primitive functions that extend the way they perform on scalars to higher order arrays.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Identity</td>
<td>(+ (number)) = (number)</td>
<td>(+4) = 4</td>
</tr>
<tr>
<td>-</td>
<td>Negation</td>
<td>(- (number)) = 0 - (number)</td>
<td>(-4) = -4</td>
</tr>
<tr>
<td>×</td>
<td>Signum</td>
<td>(× (number)) = ((number) &gt; 0) - (number) &lt; 0</td>
<td>(× -4) = -1</td>
</tr>
<tr>
<td>÷</td>
<td>Reciprocal</td>
<td>(÷ (number)) = 1 ÷ (number)</td>
<td>(÷ 4) = 0.25</td>
</tr>
<tr>
<td>~</td>
<td>Not</td>
<td>(~ (number)) = 1 - (number)</td>
<td>(~ 0 1) = 1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute Value</td>
<td>(number)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Floor</td>
<td>(number) - 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ceiling</td>
<td>(number) + 1</td>
</tr>
<tr>
<td>?</td>
<td>Random</td>
<td>A random choice from ? (number)</td>
<td>(? 10) = 7</td>
</tr>
<tr>
<td>*</td>
<td>Exponential</td>
<td>(2.71828...) * (number)</td>
<td>(* 1) = 2.71828</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Natural Logarithm</td>
<td>(2.71828...) * (number)</td>
</tr>
<tr>
<td>o</td>
<td>Pi-times</td>
<td>(3.14159...) * (number)</td>
<td>(01) = 3.14159</td>
</tr>
<tr>
<td>!</td>
<td>Factorial</td>
<td>x / ! (number)</td>
<td>(! 3) = 6</td>
</tr>
</tbody>
</table>
"Mixed functions" are those APL primitive functions which have certain requirements for the arrays they use as arguments (and, hence, do not extend the way they perform on scalars to higher order arrays).

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION/SYNTAX</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Catenation</td>
<td>Chaining: (array),(array)</td>
<td>'AB','CDE'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABCDE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iota</td>
<td>Index generator: ( \iota (\text{scalar}) )</td>
<td>( 1 ) ( 2 ) ( 3 ) ( 4 )</td>
</tr>
<tr>
<td></td>
<td>Indexing</td>
<td>Selection of specified elements: ( [\cdot] ) vector ( [\cdot] ) array ( [\cdot] ) ( [\cdot] ) ( [\cdot] )</td>
<td>( 2 ) ( 3 ) ( 5 ) ( 7 ) ( [3] ) ( 5 ) ( (2 ) ( 3 ) ( 6 ) ( [2] ) ( 2 ) ( 2 ) ( 3 ) ( 5 ) ( 7 ) ( [2;3] ) ( 21 ) ( 22 ) ( 23 ) ( 24 )</td>
</tr>
<tr>
<td>( \rho )</td>
<td>Rho</td>
<td>Size of array: ( \rho (\text{array}) )</td>
<td>( 4 ) ( 2 ) ( 3 ) ( 5 ) ( 7 )</td>
</tr>
<tr>
<td>( / )</td>
<td>Compression</td>
<td>Selection of specified elements: (logical vector)/(array)</td>
<td>( 0 ) ( 1 ) ( 1 ) ( 0 ) ( 2 ) ( 3 ) ( 5 ) ( 7 ) ( 3 ) ( 5 )</td>
</tr>
<tr>
<td>( \epsilon )</td>
<td>Membership</td>
<td>Result is 1 if element(s) on left are found on right; otherwise 0: ( (\text{array}) \epsilon (\text{array}) )</td>
<td>( 3 \epsilon 2 ) ( 3 ) ( 5 ) ( 7 ) ( 1 )</td>
</tr>
<tr>
<td>( \dagger )</td>
<td>Take</td>
<td>Take first (+) or last (-) elements: (vector)( \dagger (\text{array}) )</td>
<td>( 3 \dagger 2 ) ( 3 ) ( 5 ) ( 7 ) ( 2 ) ( 3 ) ( 5 )</td>
</tr>
<tr>
<td>( \dagger )</td>
<td>Drop</td>
<td>Drop first (+) or last (-) elements: (vector)( \dagger (\text{array}) )</td>
<td>( 3 \dagger 2 ) ( 3 ) ( 5 ) ( 7 ) ( 7 )</td>
</tr>
<tr>
<td>( ? )</td>
<td>Deal</td>
<td>(scalar) unique random integers from ( \iota (\text{scalar}) ): (scalar) ( ? (\text{scalar}) )</td>
<td>( 5 ? 5 ) ( 3 ) ( 1 ) ( 5 ) ( 2 ) ( 4 )</td>
</tr>
</tbody>
</table>

-- Continued --
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION/SYNTAX</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>Grade-up</td>
<td>Result is the permutation integers which order a vector</td>
<td>@30 20 40 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ascending:</td>
<td>4 2 1 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-descending:</td>
<td>¥30 20 40 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>¥(vector)</td>
<td>3 1 2 4</td>
</tr>
<tr>
<td>¥</td>
<td>Grade-down</td>
<td>Result is the permutation integers which order a vector</td>
<td>¥30 20 40 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-ascending:</td>
<td>4 2 1 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-descending:</td>
<td>¥30 20 40 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>¥(vector)</td>
<td>3 1 2 4</td>
</tr>
<tr>
<td>i</td>
<td>Index-of</td>
<td>Result is least indices of (array) in (vector)</td>
<td>'ABCDEF'&quot;AX'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(vector)i (array)</td>
<td>1 6</td>
</tr>
<tr>
<td>ρ</td>
<td>Restructure</td>
<td>Puts (array) in new structure (vector) ρ (array)</td>
<td>2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(vector)</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>,</td>
<td>Ravel</td>
<td>Strings out (array) into a vector , (array)</td>
<td>,2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(vector)</td>
<td>1 2 3 4 5 6</td>
</tr>
<tr>
<td>φ</td>
<td>Reversal</td>
<td>Reverses elements of an (array) about axis I</td>
<td>φ[2]2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>φ[I] (array)</td>
<td>3 2 1 6 5 4</td>
</tr>
<tr>
<td>φ</td>
<td>Rotate</td>
<td>Revolves specified numbers of elements of an (array)</td>
<td>1 −1φ[2]2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(array)φ[I] (array)</td>
<td>2 3 1 6 4 5</td>
</tr>
<tr>
<td>⊂</td>
<td>Transpose</td>
<td>Reverses order of axes ⊂(array)</td>
<td>⊂2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(array)</td>
<td>1 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 6</td>
</tr>
<tr>
<td>⊃</td>
<td>Dyadic Transpose</td>
<td>Axis I of (array) becomes axis (vector)[I] of result</td>
<td>1 1⊃2 3ρ16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(vector) ⊃(array)</td>
<td>1 5</td>
</tr>
<tr>
<td>⊓</td>
<td>Matrix Inverse</td>
<td>Result is inverse of matrix ⊓(matrix)</td>
<td>M+.× M−2 2'2ρ10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 1</td>
</tr>
<tr>
<td>⊔</td>
<td>Matrix Divide</td>
<td>Result is</td>
<td>Solution to simultaneous linear equations with coefficients B and constants A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(⊓(matrix B))+.×(matrix A)</td>
<td>(matrix A) ⊓(matrix B)</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>NAME</td>
<td>DEFINITION/SYNTAX</td>
<td>EXAMPLE</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>↓</td>
<td>Base-value</td>
<td>Decoding base-10 value of right (array) in base system of left (array) ↓ (array)</td>
<td>24 60 60↓1 2 3 3723</td>
</tr>
<tr>
<td>↑</td>
<td>Representation</td>
<td>Encoding right (array) in base system of left (array) ↑ (array)</td>
<td>24 60 60↑3723 1 2 3</td>
</tr>
<tr>
<td>*</td>
<td>Execute</td>
<td>Removes quote marks and evaluates (vector) * (vector)</td>
<td>*'3+4' 7</td>
</tr>
<tr>
<td>‭</td>
<td>Format</td>
<td>Displays (array) as a literal ‭ (array)</td>
<td>‭15 1 2 3 4 5</td>
</tr>
<tr>
<td>‭</td>
<td>Dyadic Format</td>
<td>Displays (array) as a literal with (vector)[1] column spacing and (vector) [2] significant digits, (vector) ‭ (array)</td>
<td>6 1 ‭2 3 4 2.0 3.0 4.0</td>
</tr>
<tr>
<td>\</td>
<td>Expansion</td>
<td>Expansion of (array) by (logical vector) \ (array)</td>
<td>1 0 1 0 'APL' apl</td>
</tr>
</tbody>
</table>
SUMMARY OF OPERATORS

An APL "operator" requires a function or functions (given as argument(s)) to apply to arrays. The functions must be scalar dyadic functions.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION/SYNTAX</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Reduction</td>
<td>The result is obtained by inserting the (dyadic function) between the elements of the (array) along a specified axis</td>
<td>+/[1]2 3 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(function)/[I] (array)</td>
<td>5 7 9</td>
</tr>
<tr>
<td>.</td>
<td>Outer Product</td>
<td>The result is an array obtained by applying the (dyadic function) to every pair of elements in the (arrays) given</td>
<td>(15) . 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(array). (function)(array)</td>
<td>1 2 3 4 5</td>
</tr>
<tr>
<td></td>
<td>Inner Product</td>
<td>The result is an array obtained by reducing (/) the left (dyadic function) over the result of the right (dyadic function) applied to rows of the left (array) and columns of the right (array)</td>
<td>(3 2 16) .: x2 3 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(array)(function). (function)(array)</td>
<td>9 12 15</td>
</tr>
<tr>
<td>.</td>
<td>Scan</td>
<td>The result is an array of the same size as the given (array) where each element is obtained by reducing (/) the elements up to and including it along a specified axis [I]</td>
<td>+[2]2 3 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(function)[I](array)</td>
<td>1 3 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 9 15</td>
</tr>
</tbody>
</table>
SUMMARY OF COMMANDS

An APL command causes the computer to carry out some action which has an effect on programming activity.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>NAME</th>
<th>DEFINITION/SYNTAX</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>←</td>
<td>Assignment</td>
<td>Gives a name to some data.</td>
<td>SERIES+9000 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(name)←(data)</td>
<td>NAME+&quot;HAL&quot;</td>
</tr>
<tr>
<td>→</td>
<td>Branch</td>
<td>Changes order of execution of statements in a program.</td>
<td>→(COUNTER)/7 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→(line number)</td>
<td>→4×1A=0 or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or</td>
<td>→END</td>
</tr>
<tr>
<td>∆</td>
<td>Trace</td>
<td>The result of each (line number) in (program name) is displayed as the program is executed.</td>
<td>T∆SOLVE+1 4 5 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program name ∆←(line numbers)</td>
<td></td>
</tr>
<tr>
<td>∞</td>
<td>Stop</td>
<td>The (program) automatically halts at each (line number).</td>
<td>S∆SOLVE←3 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>program name ∆+(line numbers)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quad</td>
<td>For input: □ in an expression</td>
<td>□: 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For output: □←(expression)</td>
<td>□+N</td>
</tr>
<tr>
<td></td>
<td>Quote-Quad</td>
<td>For literal input: ▼ in an expression</td>
<td>L←▼</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For output (without a carriage return): ▼+(expression)</td>
<td>WORDS ▼+L</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(position of type ball)</td>
</tr>
</tbody>
</table>

SUMMARY OF APL
The syntax of an APL program determines how it may be used. Syntax may be dyadic (2 arguments), monadic (1 argument), or nyladic (0 arguments) and may have an explicit result or no explicit result. Examples of these six different syntaxes are shown in the program definitions below:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>WITH EXPLICIT RESULT</th>
<th>NO EXPLICIT RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DYADIC</td>
<td>$\text{VALUE}+X \text{POLY} C [1] $\text{VALUE}+/X\times C \times \rho C [1] $\text{H BASEBALL AB} [1] $\text{YOUR BATTING AVERAGE} [2] $\text{H} \div \text{AB} [2] $ H</td>
<td>$</td>
</tr>
<tr>
<td>MONADIC</td>
<td>$\text{X} \div \text{AVERAGE} X [1] $X \div (+/X) \div \rho X [1] $\text{AREA S} [1] $\text{AREA OF SQUARE S IS} [2] $\text{S} \times 2 [2] $ S \times 2 [2] $</td>
<td>$</td>
</tr>
<tr>
<td>NYLADIC</td>
<td>$\text{VALUE}+\text{PI} [1] $\text{VALUE} \div \text{01} [1] $\text{VROLL} [1] $\text{THE DICE ARE} [2] $\text{?6 6} [2] $</td>
<td>$</td>
</tr>
</tbody>
</table>
SUMMARY OF FUNDAMENTAL PROGRAMMING CONCEPTS

<table>
<thead>
<tr>
<th>CONCEPT</th>
<th>EXPLANATION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Constants: numerical values or literal values.</td>
<td>3.18</td>
</tr>
<tr>
<td></td>
<td>'ABC'</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>A specific computational operation: &quot;monadic&quot; (one argument) or &quot;dyadic&quot; (two arguments) or both.</td>
<td>÷8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3+4</td>
</tr>
<tr>
<td>Variable</td>
<td>An entity, with a name, containing data which may be changed</td>
<td>N</td>
</tr>
<tr>
<td>Command</td>
<td>An explicit order which causes the computer to take some action.</td>
<td>N+5</td>
</tr>
<tr>
<td>Expression</td>
<td>A combination of data and function(s) or command(s) or program(s).</td>
<td>3×4+5</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Giving the value resulting from substituting values for variables, executing programs, and performing functions (rightmost first) in an expression.</td>
<td>3×4+5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
</tr>
<tr>
<td>Error Report</td>
<td>Brief diagnostic information about the type and location of the cause for failure of an expression to be evaluated.</td>
<td>2 4×1 3 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LENGTH ERROR</td>
</tr>
<tr>
<td>Array</td>
<td>Rectangular-structured data: scalar, vector, matrix, 3-array, 4-array, etc.</td>
<td>2 4 6</td>
</tr>
<tr>
<td>Parallel</td>
<td>Use of functions on arrays in an element-by-element fashion.</td>
<td>2 4 6×1 3 5</td>
</tr>
<tr>
<td>Processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program</td>
<td>An ordered sequence of expressions.</td>
<td></td>
</tr>
<tr>
<td>Definition</td>
<td>&quot;Writing&quot; a program. (Entering a program in the computer.)</td>
<td>VZ←MEAN X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td>&quot;Running&quot; a program. (The computer evaluating expressions in a program, line-by-line.)</td>
<td>Z+(+/X)÷ρXv</td>
</tr>
<tr>
<td>Execution</td>
<td></td>
<td>MEAN 70 75 95</td>
</tr>
<tr>
<td>Result</td>
<td>The &quot;answer&quot;. (The consequences of executing a program.)</td>
<td>80</td>
</tr>
<tr>
<td>Sub-program</td>
<td>Programs which are used in expressions within other programs.</td>
<td>VZ←VARIANCE X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z←MEAN(X-MEAN X)×2v</td>
</tr>
</tbody>
</table>

-- Continued --
<table>
<thead>
<tr>
<th>CONCEPT</th>
<th>EXPLANATION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursion</td>
<td>A program using its own name in its definition. (A program executing itself repeatedly.)</td>
<td>( \text{SUM} \cdot \text{GAUSS} ) [N]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1] ( \text{SUM} = 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2] ( \rightarrow (N \cdot 0) / 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3] ( \text{SUM} + N \cdot \text{GAUSS} ) [N - 1]</td>
</tr>
<tr>
<td>Iteration</td>
<td>A program executing certain expressions repeatedly--in a &quot;loop.&quot;</td>
<td>( \text{V} ) ( \text{TRIANGULAR} ) [N \cdot I]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[1] ( I \cdot 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[2] ( I \cdot I + 1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[3] ( \text{PRINT} : + / \cdot I )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[4] ( \rightarrow (I &lt; N) / 2 )</td>
</tr>
<tr>
<td>Names</td>
<td>Identification of programs and variables (beginning with an alphabetic letter):</td>
<td>( N ) and ( I ) and ( \text{PRINT} ) in ( \text{TRIANGULAR} ) above</td>
</tr>
<tr>
<td>Local</td>
<td>- variable names within a program only</td>
<td>( N \cdot 5 )</td>
</tr>
<tr>
<td>Global</td>
<td>- variable names within the entire active workspace</td>
<td></td>
</tr>
<tr>
<td>Workspace</td>
<td>The working area in the computer available for (disk) storage of programs and variables.</td>
<td>( \text{CLEAR} ) ) ( \text{CLEAR WS} )</td>
</tr>
<tr>
<td>Suspension</td>
<td>The condition of a program after encountering an error in one of its expressions: partially completed, &quot;suspended&quot; on a particular line.</td>
<td>( \text{TRIANGULAR} ) 'NUMBER' 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{SYNTAX ERROR} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{TRIANGULAR}[4] ) ( \rightarrow (I &lt; N) / 2 )</td>
</tr>
<tr>
<td>Debugging</td>
<td>Any methods of pinpointing errors (&quot;bugs&quot;) in programs and fixing them.</td>
<td>( \text{from above} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( N \cdot 5 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \rightarrow 4 )</td>
</tr>
<tr>
<td>Interactive Program</td>
<td>A program which interacts with the user, i.e. typically prints output, accepts input, alternatingly.</td>
<td>( \text{DRILL} ) ( \text{WHAT IS} 3 \times 4 ? )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \square : ) 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{WHAT IS} 9 \times 7 ? )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \square : ) 63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>etc.</td>
</tr>
<tr>
<td>Simulation</td>
<td>A program which simulates some real-world phenomenon via a mathematical/computational model--usually a simplification and possibly a distortion.</td>
<td>( \text{VTEMPER THRESHOLD} ) ( \text{EMOTION} \cdot 0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{NEW: EVENT} \cdot ? 1.0 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{EMOTION} \cdot \text{EVENT} + \text{EMOTION} \cdot 2 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \rightarrow (\text{EMOTION} &gt; \text{THRESHOLD}) / \text{MAD} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \rightarrow \text{NEW} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{MAD} : ' **!?!*!!' )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \rightarrow 1 )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{V} )</td>
</tr>
</tbody>
</table>
SUMMARY OF ERROR REPORTS

Error reports give general diagnostic information about the type and location of errors in expressions.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>INTERPRETATION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNTAX</td>
<td>Faulty syntax in an expression, i.e. a function or program used without value(s) in the proper place</td>
<td>$4+3x$ SYNTAX ERROR</td>
</tr>
<tr>
<td>VALUE</td>
<td>A name used without having been assigned a value</td>
<td>$8\times X$ VALUE ERROR</td>
</tr>
<tr>
<td>INDEX</td>
<td>Improper indexing, e.g. an index using a negative number, a non-integer, or an integer larger than the size of an array</td>
<td>'ABCD'[5] INDEX ERROR</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>A value outside of the domain of values used with a particular function</td>
<td>$5\div 0$ DOMAIN ERROR</td>
</tr>
<tr>
<td>LENGTH</td>
<td>The size (length) of one array does not match the size of the other array used with a function</td>
<td>$2 \times 3 \times 2 \times 3 \times 4$ LENGTH ERROR</td>
</tr>
<tr>
<td>DEFN</td>
<td>Improper attempt at defining or editing a program.</td>
<td>$\forall A B C D$ DEFN ERROR</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>Improper formation of a character</td>
<td>$\forall A B C D'$ CHARACTER ERROR</td>
</tr>
<tr>
<td>RANK</td>
<td>A function used with value(s) of the wrong rank</td>
<td>$14 5 6$ RANK ERROR</td>
</tr>
<tr>
<td>LABEL</td>
<td>Improper use of line labels in a program</td>
<td>$\forall$ START: LABEL ERROR</td>
</tr>
<tr>
<td>WS FULL</td>
<td>Workspace capacity too small to complete computation</td>
<td>(11000)$.\times 11000 WS FULL</td>
</tr>
</tbody>
</table>
SUMMARY OF EDITING PROCEDURES

Editing procedures are used to define, refine, or change a program.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NOTATION</th>
<th>EFFECT</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>( \text{program name} )</td>
<td>Change from command execution mode to program definition mode.</td>
<td>( \text{\textless PROGRAM} \ [1] )</td>
</tr>
<tr>
<td>Display</td>
<td>( \text{program line number} )</td>
<td>Display (line number) or Display whole program if (line number) is omitted.</td>
<td>( \text{\textless PROGRAM}[3]) or ( \text{\textless PROGRAM}[] )</td>
</tr>
<tr>
<td>Override</td>
<td>( \text{program line number} ) ( \text{(expression)} )</td>
<td>Replace an expression on a given (line number) with a new (expression) in program (name).</td>
<td>( \text{\textless PROGRAM}[5]\text{B+1} )</td>
</tr>
<tr>
<td>Add</td>
<td>( \text{program name} )</td>
<td>Add new line(s) on to a previously defined program (name).</td>
<td>( \text{\textless PROGRAM} \ [7] )</td>
</tr>
<tr>
<td>Insert</td>
<td>( \text{program line number} ) ( \text{(decimal line number)} ) ( \text{(expression)} )</td>
<td>Insert a new line between (decimal line number) and (decimal line number) in program (name).</td>
<td>( \text{\textless PROGRAM}[2.5] )</td>
</tr>
<tr>
<td>Delete</td>
<td>( \text{program line number} ) ( \text{ATTN} )</td>
<td>Remove (line) from program (name).</td>
<td>( \text{\textless PROGRAM}[4] \text{[4] ATTN} )</td>
</tr>
<tr>
<td>Change Header</td>
<td>( \text{program <a href="header">0</a>} )</td>
<td>Give program (name) a new(header).</td>
<td>( \text{\textless PROGRAM}[0]\text{NEW} )</td>
</tr>
<tr>
<td>Character</td>
<td>( \text{program line number } ) ( \text{(spaces)} )</td>
<td>Prepare for changing specific characters on a (line) in a program (name) by displaying the line and spacing the type ball over a certain number of (spaces). Then / is used to strike out characters, and numbers insert spaces in front of characters.</td>
<td>( \text{\textless PROGRAM}[2\text{D}7] )</td>
</tr>
</tbody>
</table>
### LIST OF SYSTEM COMMANDS

A "system command" is used for workspace control and library management. (A library is a collection of workspaces.)

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>DEFINITION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAVE (work-space name)</td>
<td>Store the current (workspace) on disk memory. All programs and variables are saved.</td>
<td>SAVE MYWORK MYWORK SAVED 07/08/77</td>
</tr>
<tr>
<td>LOAD (work-space name)</td>
<td>Retrieve the (workspace) from disk memory to become the active workspace.</td>
<td>LOAD GAMES GAMES SAVED 04/14/77</td>
</tr>
<tr>
<td>COPY (library number) (work-space name) (program or variable names)</td>
<td>Copy particular (programs and/or variables) from a particular (workspace) in a particular (library) into the current active workspace.</td>
<td>COPY 123456 PLOT GRAPH</td>
</tr>
<tr>
<td>FNS</td>
<td>List alphabetically the names of all defined functions in the active workspace.</td>
<td>FNS GRAPH HANGMAN NIM MOVE</td>
</tr>
<tr>
<td>VARS</td>
<td>List alphabetically the names of all global variables in the active workspace.</td>
<td>VARS A B X Y</td>
</tr>
<tr>
<td>LIB</td>
<td>List the names of workspaces in user's library.</td>
<td>LIB MYWORK GAMES</td>
</tr>
<tr>
<td>WSID</td>
<td>Workspace Identification. Result is the name of the current active workspace</td>
<td>WSID GAMES</td>
</tr>
<tr>
<td>SI</td>
<td>State Indicator. Lists all suspended programs (including &quot;pendant&quot; programs which have yet to be completed due to the suspended programs) marked with asterisks.</td>
<td>SI HANGMAN[29] * NIM[3] * MOVE[1]</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Clear the active workspace.</td>
<td>CLEAR</td>
</tr>
<tr>
<td>ERASE (program or variable names)</td>
<td>Remove a (program) or global (variable) from the active workspace.</td>
<td>ERASE GRAPH</td>
</tr>
<tr>
<td>DROP (work-space name)</td>
<td>Permanently remove the contents and name of a (workspace).</td>
<td>DROP MYWORK MYWORK DROPPED 07/08/77</td>
</tr>
</tbody>
</table>
LIST OF SYSTEM VARIABLES

A "System variable" is a special variable which contains information relevant to the computing system and which may be used in APL expressions.

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>NAME</th>
<th>DEFINITION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>□IO</td>
<td>Index Origin</td>
<td>Value is 0 or 1; used as the beginning of indices.</td>
<td>□IO+0 14</td>
</tr>
<tr>
<td>□PP</td>
<td>Printing Precision</td>
<td>Value is number of significant digits displayed in numerical output.</td>
<td>□PP+3 01</td>
</tr>
<tr>
<td>□PW</td>
<td>Printing Width</td>
<td>Value is the number of columns used in printing across the page/screen on a terminal.</td>
<td>□PW+10 1 2 3 4 5</td>
</tr>
<tr>
<td>□CT</td>
<td>Comparison Tolerance</td>
<td>Value is the number to which the difference of two numbers is compared in order to judge if they are equal.</td>
<td>□CT+.01 3.14=01 1</td>
</tr>
<tr>
<td>□LX</td>
<td>Latent Expression</td>
<td>Value is vector of characters executed immediately (using £) upon loading a workspace.</td>
<td>□LX+''HI!'' SAVE MYWORK LOAD MYWORK</td>
</tr>
<tr>
<td>□RL</td>
<td>Random Link</td>
<td>Value is used by ? to generate random numbers.</td>
<td>□RL+16807 ?10</td>
</tr>
<tr>
<td>□AI</td>
<td>Accounting Information</td>
<td>Values are: identification # computer time, connect time, keying time (milliseconds).</td>
<td>□AI 123456 25 689200 8716</td>
</tr>
<tr>
<td>□LC</td>
<td>Line Counter</td>
<td>Values are statement numbers of programs being executed (esp. suspended programs).</td>
<td>□LC 18</td>
</tr>
<tr>
<td>□TS</td>
<td>Time Stamp</td>
<td>Values are: year, month, day, hour, minute, second, and millisecond of current time.</td>
<td>□TS 1977 12 24 23 59 59 9</td>
</tr>
<tr>
<td>□TT</td>
<td>Terminal Type</td>
<td>Value is: 1 - selectric; 2 - PTTC/BCD; 3 - 1050; 4 - 3270.</td>
<td>□TT 1</td>
</tr>
<tr>
<td>□UL</td>
<td>User Load</td>
<td>Value is the numbers of users currently on the (time-sharing) system.</td>
<td>□UL 23</td>
</tr>
<tr>
<td>□WA</td>
<td>Working Area</td>
<td>Value is the number of bytes of storage space remaining in the current active workspace.</td>
<td>□WA 32000</td>
</tr>
</tbody>
</table>
A "system function" is a special function which affects how the computing system performs and which may be executed in APL expressions.

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>NAME</th>
<th>DEFINITION</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>□CR</td>
<td>Canonical</td>
<td>Result is literal matrix with rows of expressions from each line in a defined program, given as a (literal). □CR (literal)</td>
<td>□CR 'AVERAGE' Z+AVERAGE N Z+=(+/N):pN</td>
</tr>
<tr>
<td>□FX</td>
<td>Fix</td>
<td>Result is defined program with expressions on each line from rows of a (literal matrix). □FX (literal matrix)</td>
<td>□FX 2 11 p 'Z+AVERAGE N Z+=(+/N):pN '</td>
</tr>
<tr>
<td>□EX</td>
<td>Expunge</td>
<td>Result is erasure of program or local variable given as (literal) name. □EX (literal)</td>
<td>□EX 'AVERAGE' 1</td>
</tr>
<tr>
<td>□NL</td>
<td>Name List</td>
<td>Result is vector list of first (n) names of labels (1), variables (2), or programs (3): (n) □NL (1,2, or 3) or all names: □NL (1,2, or 3)</td>
<td>1 □NL 3 AVERAGE or □NL AVERAGE</td>
</tr>
<tr>
<td>□NC</td>
<td>Name Class</td>
<td>Result is 0 if name is unused, 1 if used as a label, 2 as a variable, 3 as a program, 4 other. □NC (name)</td>
<td>□NC 'AVERAGE' 3</td>
</tr>
<tr>
<td>□DL</td>
<td>Delay</td>
<td>Postpone execution a specified number of (seconds). □DL (seconds)</td>
<td>□DL 60</td>
</tr>
</tbody>
</table>
Appendix

ANSWERS

Contents

U-Program 1 218
U-Program 2 220
U-Program 3 222
U-Program 4 224
U-Program 5 227
U-Program 6 229
U-Program 7 230
U-Program 8 232
U-Program 9 235
U-PROGRAM 1

Page 2

'WITH SOME FOR YOU TO DO'
WITH SOME FOR YOU TO DO
4 + 8
12
7 - 3
4
5 × 20
100
100 ÷ 4
25

Page 3

2.5 + 7.1
9.6
4 - 7
3
3.0 × 5
15
100 ÷ 3
33.33333333

Page 4

A - B
3
A × B
130

Page 5

COUNTER
4
COUNTER
5
YEAR
2001

Page 6

SET
2 3 5 7
SET - 1
1 2 4 6
SET × 2
4 6 10 14

Page 7

SET × SIX
8 9 11 13 17
SET , 6
2 3 5 7 11 6

Page 8

V × W
8 0 5 35 33
W × V
8 0 5 35 33

Page 9

D, D, D, D
****
D, E, S, I, G, N, S
*ΔΔ*Δ0Δ0*Δ*Δ

Page 10

A , L , A
ABRACADABRA
S , H , O , T
CURSE YOU, RED BARON!!!

Page 11

8 = 11
0
12 = 12
1

Page 12

V < 5
1 0 1 1 0 1 0
V ≤ 5
1 1 1 1 1 1 0

APPENDIX: ANSWERS
\[ V > 5 \]
\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]
\[ V \geq 5 \]
\[
\begin{array}{ccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1
\end{array}
\]
************************

Page 13

\[ 4 \neq 4 \]

\[ 0 \]

\[ 4 \neq 7 \]

\[ 1 \]

'\[ \square \]' \[ \neq \] '\[ \square \]' \[ 0 \]

'\[ \square \]' \[ \neq \] '\[ \| \]' \[ 1 \]

'\[ B \]' \[ = \] 'ABBABA' \[ 1 \]

0 0 0 1 0 1

'\[ B \]' \[ = \] 'ABBABA' \[ 0 1 1 0 1 0 \]

************************

Page 14

10 \[ \uparrow \] \[ 8 \]

10

12 \[ \uparrow \] \[ 8 \]

12

8 \[ \uparrow \] \[ 12 \]

8

************************

Page 15

\[ V \geq 5 \]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]

\[ V > 5 \]

\[
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]

************************

Page 16

\[ 3 \mid 9 \mid 10 \mid 11 \]

0 1 2

0 0 0 1 0 1 0 1

0 1 2 3 0 1 2 3 0

5 \mid \mid 6 \mid 4 \mid 2 \mid 0 \mid 2 \mid 4 \mid 6

4 1 3 0 2 4 1

************************

Page 18-Problems

\[ T + S \]

7.2 4

'T + S' \[ T + S \]

\[ T - S \]

\[ -0.8 \]

\[ T \times S \]

12.8 \[ -12 \]

\[ T \times S \]

0.8 \[ -3 \]

APPENDIX: ANSWERS 219
U-PROGRAM 2

***************
Page 20

AREA
THE AREA IS
81
***************

Page 21

AREA
THE AREAS ARE
9 16 25 64
***************

Page 22

BASEBALL
THIS PROGRAM COMPUTES BATTING AVERAGE.
0.315
***************

Page 23

A
42
***************

Page 24

TRIANGLE
2 8 18 32 50
***************

Page 25

15
1 2 3 4 5

13
1 2 3

^1
DOMAIN ERROR
^1
^13.5
DOMAIN ERROR
^13.5
^1

Page 26

V[3]
5

V[5]
INDEX ERROR
W[5]
^1

Page 27

W[2]
9

W[3]
2

W[2 + 3]
7

11

W[5.5]
DOMAIN ERROR
W[5.5]
^1

W[6]
1

Page 28

pY
8

220 APPENDIX: ANSWERS
p'ABCD'
4

p'SHAKESPEARE'
24

**************************
Page 29

L[1]
T

L[2 4 1], ' ', L[1 4 3 5], 'S'
RAT TAILS

L
TWIST
pL
5

**************************
Page 30

5 + 9 + 2 + 0 + 7 + 1
24

+/19
45

**************************
Page 31

SUM + N
7

SUM
21

N
3

**************************
Page 32

AVERAGE W
4

W
5 9 2 0 7 1

AVERAGE W
5

**************************
Page 34- Problems

REVIEW
THE ANSWERS ARE
9
6
1 2 3 4 5 6 7 8 9
5
11
8
41
45
13
2 3 5 7 11 13
41
4
66

APPENDIX: ANSWERS 221
**U-PROGRAM 3**

***************

**Page 36**

\((6 \times 4) + 5\)

29

\(6 \times (4 + 5)\)

54

\(6 \times 4 + 5\)

54

\(6 + 4 \times 5\)

26

\(6 + (4 \times 5)\)

26

***************

**Page 37**

\((2 \times 3 + 5 \times 4) = (2 \times (3 + (5 \times 4)))\)

1

Z1

9

***************

**Page 38**

T

2

S

8

R

3

Q

9

***************

**Page 39**

\(5 + 9 + 2 + 6 + 7 + 1\)

30

\(5 + (9 + (2 + (6 + (7 + 1))))\)

30

SUM

1

SUM

8

SUM

14

SUM

16

SUM

25

SUM

30

***************

**Page 40**

\(5 \times 9 \times 2 \times 6 \times 7 \times 1\)

3780

MAX

7

MAX

7

***************

APPENDIX: ANSWERS
Page 41

\[
\begin{align*}
\text{MAX} & \quad \text{MAX} \\
7 & \quad 9 \\
\text{MAX} & \quad \text{MAX} \\
9 & \quad 9
\end{align*}
\]

**********************

Page 41-Problems

\[
\begin{align*}
\text{L/W} & \quad \text{L/W} \\
1 & \quad 1 \\
\text{MIN} & \quad \text{MIN} \\
1 & \quad 1 \\
\text{DIFF} & \quad \text{DIFF} \\
6 & \quad 6 \\
\text{DIFF} & \quad \text{DIFF} \\
0 & \quad 0 \\
\text{DIFF} & \quad \text{DIFF} \\
2 & \quad 2 \\
\text{DIFF} & \quad \text{DIFF} \\
7 & \quad 7 \\
\text{DIFF} & \quad \text{DIFF} \\
-2 & \quad -2
\end{align*}
\]

Page 43

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
4 & \quad 4 \\
6 & \quad 6 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 \times 2 & \quad (14) \times 2 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

Page 42

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
-16 & \quad -16 \\
-3 & \quad -3 \\
(+/S[1\ 3\ 5]) & \quad (+/S[2\ 4\ 6]) \\
-3 & \quad -3 \\
+/16 & \quad +/16
\end{align*}
\]

Page 45-Problems

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8 \\
9 & \quad 9
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
2 & \quad 2 \\
4 & \quad 4 \\
6 & \quad 6 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]

\[
\begin{align*}
\text{DIFF} & \quad \text{DIFF} \\
14 & \quad 14 \\
1 & \quad 1 \\
2 & \quad 2 \\
3 & \quad 3 \\
4 & \quad 4 \\
5 & \quad 5 \\
6 & \quad 6 \\
7 & \quad 7 \\
8 & \quad 8
\end{align*}
\]
U-PROGRAM 4

Page 48

1\wedge 1
1

1\wedge 0
0

0\wedge 1
0

0\wedge 0
0

1\vee 1
1

1\vee 0
1

0\vee 1
1

0\vee 0
0

~1
0

~0
1

Page 49

~L \wedge K
0 1 1 1

(~L) \wedge \neg K
0 0 0 1

+/\neg(L \wedge K) \wedge L \vee \neg L = K
3

Page 50

\wedge/K
0

\vee/K
1

Page 51

\wedge/L \vee K
0

\vee/L \wedge K
1

Page 52

Q \times P
2 0 5 0 11 13 0 0

Page 53

K / 6 2 8 4
6 8

K / 'FLIP'

FI

1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 /'STOP THE RECORD'

STEREO

Page 54

1 0 / 3 5
3

0 1 / 3 5
5

0 0 / 3 5
***************
Page 55

POW

4
16
64
256

***************
Page 59

POW

5
25
125
625

***************
Page 62

2 POWER 3

8

***************
Page 65

6 * 2
36

7 8 9 10 * 1 2 3 4
7 64 729 10000
3 * 3 * 3
27
3 * 4
81
3 * 3 * 3 * 3
81

***************
Page 66

9 *.5
3

-16 *.5

DOMAIN ERROR

~16*0.5
^ 

-8 * 1 + 3
-2
2 * -1
0.5
0 * 0
1

***************
Page 67

AB 8
8

AB -8
8

ABS 11
11

***************
Page 68

| -3 * -3 |
9

| -3 * -3 |
9

***************
Page 69

5 RES 13
3

***************
Page 70

5 RES 13
3

5 | 13
3

3.14 - 1 RES 3.14
3

FLOOR 3.14
3

CEILING 3.14
4

***************
Page 71

| 8.0 8.3 8.6 8.9 9.2 9.5 |
8 8 8 8 9 9
| 8.0 8.3 8.6 8.9 9.2 9.5 |
8 9 9 9 10 10

***************
Page 72

ROUND 3.14
3

ROUND 3.6
4

ROUND ~2.55
-3

ROUND ~2.0904
-2

(10 ^ -4) * | 0.5 + X * 10 ^ 4 |
1.6667

APPENDIX: ANSWERS 225
REVIEW

\[(4 = 4) \land 5 = 5\]

\[1\]

\[(3 \geq 4) \lor 5 \neq 5\]

\[0\]

\[(-\land/\text{LOGICAL}) = \lor/\neg\text{LOGICAL}\]

\[1\]

\[2|+/\text{LOGICAL}\]

\[1\]

\[2***/\text{LOGICAL}\]

\[8\]

\[|+/\text{LOGICAL}\]

\[3\]

\[|-/\text{LOGICAL}\]

\[1\]

\[\Box+P+2\]

\[2\]

\[\Box=X+20*3\]

\[6.666666667\]

\[(10*-P)\times1.5+X*10*P\]

\[6.67\]

\[\lor OFF*X ROUND P\]

\[1\]

\[OFF+(10*-P)\times10.5+X*10*P\]

\[\lor\]

\[\lor Z+L \text{MIN R}\]

\[1\]

\[+(L<R)/4\]

\[2\]

\[Z=R\]

\[3\]

\[\rightarrow 0\]

\[4\]

\[Z+L\neg\]

\[T\Delta\text{MIN}+14\]

\[P=1.667 \text{ MIN 2}\]

\[\text{MIN}[1] 4\]

\[\text{MIN}[4] 1.667\]

\[S \text{ MIN R MAX } T[S]\]

\[3\]

\[T * S\]

\[1 2 \ 4\]

\[S * T\]

\[3 9 \ 81\]

\[(S * 2) \lor 2 * S\]

\[9\]

\[((S+1) * 2) = (S * 2) + (2 * S) + 1\]

\[1\]
U-PROGRAM 5

'B' ∈ VOWELS
0

'COMPUTER' ∈ VOWELS
0 1 0 0 1 0 1 0

Page 83

v/ 'LINGO' ∈ VOWELS
1

VOWELCHECKER 'CONSONANTS'
1

VOWELCHECKER 'WHYZZ'
0

Page 85

4 + W
5 9 2 6

W = 6 + W
1 1 1 1 1

-3 + W
6 7 1

-5 + W
9 2 6 7 1

-8 + W
0 0 5 9 2 6 7 1

Page 86

3 + W
6 7 1
***************

Page 89

13 ? 52
16 5 12 37 46 13 31 17 11 1 27 48 43

13 ? 13
4 3 13 10 11 1 6 9 2 12 7 5 8

14 ? 13
DOMAIN ERROR
14?13

***************

Page 90

I
3 1 5 2 4

[] + PER + \#I
1 2 3 4 5

[] + PER + ...\[I]
3 1 5 2 4

[] + PER + ...\[I]
5 3 4 1 2

[] + PER + ...\[I]
4 5 2 3 1

[] + PER + ...\[I]
2 4 1 5 3

***************

Page 91

\$D2
3 5 4 1 6 2

D2[\$D2]
9 7 6 2 0 $2

\#6 5 7 8 9
5 2 1 3 4

S
TUNPEEN

S[\#N]
NEPTUNE

S[\#N]
NEPTUNE

***************

Page 92

D2[\$D2]
9 7 6 2 0 $2

\#6 5 7 8 9
5 2 1 3 4

S
TUNPEEN

***************

Page 93

ALPHABET \ 'MAN'
13 1 14

ALPHABET[18 15 2 9 14]
ROBIN

SORT V[(pV)?pV]
1 + pALPHABET

***************

Page 94

LSORT 'SLOT'
LOST

***************

Page 95

&6 9 ~2 2 0 7
2 4 3 0 5 1

&6 9 ~2 2 0 7
1 5 0 3 4 2

18
1 2 3 4 5 6 7 8

'ZERO'[2]
E
**U-PROGRAM 6**

***************

**Page 98**

B

9

N

34

***************

**Page 99**

\[ 5 \times 8 \iff A + 2 \]

\[ \square: \]

7

45

***************

**Page 100**

B

ENTER

C

ANY

\[ pC \]

3

B \cdot C \cdot A

ENTER ANY LITERALS

***************

**Page 101**

X

ACETYLENE

***************

**Page 108**

\[ \square: \]

END

VALUE ERROR

END

\[ ^{\wedge} \]

***************

**Page 109**

\[ \square: \]

DRILL

\[ 14 \times 75 \]

\[ \square: \]

1050

\[ 46 \times 53 \]

\[ \square: \]

2438

\[ 22 \times 5 \]

\[ \square: \]

110

\[ 68 \times 68 \]

\[ \square: \]

4624

\[ 93 \times 38 \]

\[ \square: \]

3534

CONGRATULATIONS! WOULD YOU LIKE 5 MORE?

ENTER Y FOR YES, N FOR NO.

YES

52 \times 38

\[ \square: \]

STOP

***************

**Page 113**

\[ \backslash \text{DRILL[18]} \rightarrow (\backslash '!YES' \& 0) / \backslash V \]

\[ \text{DRILL} \]

19 \times 36

\[ \square: \]

STOP

***************

**Page 116**

TEMPE

HOW DO YOU FEEL ABOUT ME?

\[ \square: \]

8

\[ \square: \]

6

\[ \square: \]

4

\[ \square: \]

\

***************

**Page 117**

\[ ^{\wedge} \text{NO LIMIT} \]

APPENDIX: ANSWERS 229
AND MILES TO GO BEFORE I SLEEP
AND MILES TO GO BEFORE I SLEEP
AND MILES TO GO BEFORE I SLEEP

TO
BE
OR
NOT
TO
BE

GOOD
PLAY
BILL

 MATRIX-2
  7  3 -2  4
  0  2  9  1
 14 6 18  5

 MATRIX
  9  6  6  6
  6 11  6
 16 8 20  7

 MATRIX=3
 0 0 0 0
 0 0 0 1
 0 0 0 0

3eMATRIX
 1

 MATRIX€3
 0 0 0 0
 0 0 0 1
 0 0 0 0

 *******************
 Page 127

 Matrix
 8

 MATRIX[3;]
 16 8 20  7

 MATRIX[;2]
 5 4 8

 *******************
 Page 128

 Matrix
 3 4 1

 MATRIX[3 2 3;3]
 20 11 20

********************
Page 124

 3 30p AND MILES TO GO BEFORE I SLEEP
 3 30p TO BE OR NOT

Page 125

 \rho, \M

 M×3
 3  6  9 12
15 18 21 24
27 30 33 36

(\cdot, \M) = (\times, \rho\M)\rho\M

1

Page 128

 Matrix
 9 5 0 6
 2 4 11 3
16 8 20  7

MH

 Matrix
 3 4

 Matrix
 9 5 0 6 2 4 11 3 16 8 20  7

OH! OH! OH!
\[
\text{MATRIX}[1 2 3; 1 2 3] = \text{MATRIX}[2;2]
\]

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]

\[
\text{MATRIX}[I[I[1];I[2]]] = \text{MATRIX}[;I[2]]
\]

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1
\end{bmatrix}
\]

*******
Page 129

\[
+/\text{MAT}
\]

6 15

\[
-#/\text{MAT}
\]

\[
-3 -3 -3
\]

\[
(+/+/\text{MAT}) = --/\text{MAT}
\]

\[
0
\]

\[
(1 \times 3 \times 5 \times 2 \times 4 \times 6) = 4/\text{MAT}
\]

1

*******
Page 130

\[
\times/\text{MAT}
\]

6 120

\[
-#/\text{MAT}
\]

\[
-3 -3 -3
\]

\[
(+/+/\text{MAT}) = --/\text{MAT}
\]

\[
0
\]

\[
(1 \times 3 \times 5 \times 2 \times 4 \times 6) = 4/\text{MAT}
\]

1

*******
Page 131

\[
\text{LHM}
\]

SOLD
OHIO
U-PROGRAM 8

Page 140

\[ \phi' \text{NOSLIW}' \]

WILSON

\[ \phi' \text{DOCNOTEIDISSENTAFASTNEVERPREVENTSAFATNESSIDIETONCOD}' \]

\[ \text{DOCNOTEIDISSENTAFASTNEVERPREVENTSAFATNESSIDIETONCOD} \]

\[ \phi \text{M} \]

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

Page 141

\[ (\rho \text{M}) = \phi \text{M} \]

\[
\begin{array}{cccccc}
1 & 1 \\
\end{array}
\]

\[ 4 \rho \text{'FOEANDICLEN'} \]

FAIL
ONCE
EDEN

\[ 1\phi \text{'TOPS'} \]

STOP

Page 142

\[ (\rho R) = \rho B \]

\[
\begin{array}{cccc}
1 & \\
\end{array}
\]

\[ -1\phi \text{'UPCLEESAAP'} \]

APPLE
SAUCE

\[ -2\phi \text{M} \]

IBM

Page 143

\[ 2\phi \text{M} \]

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

\[ -1\phi \text{M} \]

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

Page 144

\[ 2 \ 3 \ \Phi \ 2 \ 5\rho \text{'LESTA'} \]

STALE
TALES

Page 145

\[ (\rho L) = (\rho L)[2 \ 1] \]

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[ \Phi \text{LM} \]

Page 146

\[ D[I] = SQ[I;I] \]

\[ \Phi \phi \text{LM} \]

\[ (\rho D) = (\rho) SQ \]

\[ 1 \]

Page 147

\[ \Phi \text{LM} \]

\[ ** \]

\[ \Phi \text{LM} \]

\[ ** \]

232 APPENDIX: ANSWERS
(v/-VEGETABLE)'/PEARS'

PEAS

*******************************

Page 154

[] + P + 3 4 P 6 1 2 3 0 5
6 1 2
3 0 5

[] + M + 3 4 P 1 2 12
1 2 3 4
5 6 7 8
9 10 11 12

[] + Q + P + x M
29 38 47 56
48 56 64 72

ρQ
2 4

*******************************

Page 151

P 10 4 x 3 2
38

(+V x M[;1:], +V x M[;2])
60 84

V+[M
20 19

*******************************

Page 152

v/1 0 1 A 0 0 1
1

v/0 1 0 A 0 0 1
0

v/1 0 1 A 0 0 1
1

*******************************

Page 153

(-1+pM), 1+pN
3 5

+/M[;1] x N[;1]
110

R[2;3] = +/M[2;1] x N[;3]
1

*******************************

Page 150

(10) x 10

1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100

*******************************

Page 156

[] + M + 3 4 P 1 2 12
1 2 3 4
5 6 7 8
9 10 11 12

0,M
0 1 2 3 4
0 5 6 7 8
0 9 10 11 12

0,M
0 0 0 0
1 2 3 4
5 6 7 8
9 10 11 12

*******************************

Page 157

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

M,[1]M
1 2 3 4
5 6 7 8
9 10 11 12

APPENDIX: ANSWERS 233
Page 158

\[ V_{[.5]}V \]

\[
\begin{array}{cc}
2 & 3 & 5 & 7 \\
2 & 3 & 5 & 7 \\
\end{array}
\]

\[ \rho V_{[.5]}V \]

\[
\begin{array}{c}
2 \\
4 \\
\end{array}
\]

\[ V_{[1.5]}V \]

\[
\begin{array}{cc}
2 & 2 \\
3 & 3 \\
5 & 5 \\
7 & 7 \\
\end{array}
\]

\[ \rho V_{[1.5]}V \]

\[
\begin{array}{c}
4 \\
2 \\
\end{array}
\]

Page 159

\[ {}^{2}L^{+4} 3p'ABCDEFGHJKLM' \]

\[
\begin{array}{cccc}
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\end{array}
\]

\[ {}^{2}L'{}^{*} \]

\[
\begin{array}{cccc}
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\end{array}
\]

\[ {}^{2}L'_{L} \]

\[ {}^{2}L, [2.5]L \]

Page 161

\[ {}^{2}L, [2.5]L \]

\[
\begin{array}{cccc}
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\end{array}
\]

\[ {}^{2}L, [1]L \]

\[
\begin{array}{cccc}
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\text{ABC} & \text{DEF} & \text{GHI} & \text{JKL} \\
\end{array}
\]

\[ {}^{2}C+3 3p 2 \begin{pmatrix} -1 & 5 & 1 & 2 & 1 & 4 & 0 & -1 \end{pmatrix} \]

\[
\begin{array}{cccccccc}
2 & -1 & 5 \\
1 & 2 & 1 \\
4 & 0 & -1 \\
\end{array}
\]

\[ {}^{2}B+13 0 \begin{pmatrix} 11 \end{pmatrix} \]

\[
\begin{array}{cccc}
13 & 0 & 11 \\
\end{array}
\]
U-PROGRAM 9

Page 164

+5
5

-6
-6

V + W
0 0 0 0 0 0

Page 165

- 0>−5
-1

SIGNUM B
1 -1 0 1 0 -1

B × SIGNUM B
3 4.2 0 5.8 0 9

Page 166

+3
0.3333333333

+10
0.1

Page 167

2.718281828*1
2.718281828

Page 168

*3
3

Page 169

10•10*2
2

10•1000
3

10•10000 100000 1000000 10
4 5 6 1

7•7•3
3

3•81
4

Page 170

02
6.283185307

Page 171

π-RADIANS+o+6+12
0.524 1.05 1.57 2.09 2.62 3.14 3.67 4.19 4.71 5.24 5.76 6.28
30 RADIANS
0.577  1.73  5.73E15  -1.73  -0.577  -1.74E16  0.577  1.73
5.73E15  -1.73  -0.577  -3.49E16

************************

Page 174

DEGREES 60
1.05

COSINE DEGREES 60
0.5

(DEGREES 45) x 360 + 0.2
45

************************

Page 175

RADIANS 0+4
45

30 = RADIANS DEGREES 30
1

************************

Page 176

X
3

Y
1 2 3 4 5

X MEMBER Y+2xY
0

'R' MEMBER 'WORD'
1

************************

Page 177

FAC 4
24

************************

Page 178

FAC 0
FAC 0
4

************************

Page 179

211 0 1
5

211 0 1 1 1
23
\[(1 \times 2 \times 4) + (0 \times 2 \times 3) + (1 \times 2 \times 2) + (1 \times 2 \times 1) + 1 \times 2 \times 0\]

23

\[(1 \times 5 \times 3) + (4 \times 5 \times 1) + (2 \times 5 \times 1) + 3 \times 5 \times 0\]

238

****************

Page 180

\((8 \times 2 \times 4 \times 7 \times 3) - 8 \times 2 \times 4 \times 6 \times 3\)

8

\((8 \times 2 \times 4 \times 7 \times 3) - 8 \times 2 \times 3 \times 7 \times 3\)

64

\cdot (8 \times 2 \times 4 \times 7 \times 3) - 8 \times 1 \times 1 \times 1

754

10 \times 1 \times 7 \times 7 \times 6

1776

10 \times 1 \times 4 \times 4

444

10 \times 10 \times 10 \times 1 \times 7 \times 7 \times 6

1776

****************

Page 181

\[O = R + W + . \times B\]

3782

\[O = W + (x / 1 + A), (x / 2 + A), (x / 3 + A), 1\]

86400 3600 60 1

1780 3 12 \times 15 2 6

210

24 60 60 \times 3782

1 3 2

****************

Page 182

\[(4 \times p10) \times (4 \times p10) \times 2 \times 0 \times 0 \times 1\]

2 0 0 1

2 2 2 \times 4

1 0 0

2 \times 0

0

****************

Page 183

\[2 2 2 \times 4 5 6 7 8\]

1 1 1 1 0

0 0 1 1 0

0 1 0 1 0

\((\bar{W} \times 2 \times N) - 1\)

0 0 0 0 1 1 1 1

0 0 1 1 0 0 1 1

0 1 0 1 0 1 0 1

****************

Page 184

\[TRUTH 2\]

0 0

0 1

1 0

1 1

\[pTRUTH 4\]

16 4

****************

Page 185

\[1780 3 12 DECODE 5 2 6\]

210

\[1780 3 12 ENCODE 210\]

5 2 6

****************

Page 186

\[D\]

\[VALUE ERROR\]

\[D\]

\[\wedge\]

'1 2 3 4 5' = \wedge 5

1 1 1 1 1 1 1 1 1

APPENDIX: ANSWERS 237
\[(p \downarrow 15) = p \downarrow 15'\]

0

***************

Page 187

2\downarrow 15
1.00 2.00 3.00 4.00 5.00
\[p \downarrow V \downarrow A\]
10 30

***************

Page 188

\[(+/11), (+/12), (+/13), (+/14), (+/15), (+/16), (+/17), (+/18), (+/19), (+/10)\]
1 3 10 15 21 28 36 45 55
\[\backslash 10\]
0 0 0 0 0 0 0 0 0
\[\backslash 0\]
0 0 1 0 1 1 0
0 0 1 1 1 1 1
\[\land 1 1 0 1 0 0 1\]
1 1 0 0 0 0 0

***************

Page 189

\[p R\]
4
\[+/Q\]
4
\[(pQ) = p V\]
1

L'\text BACKS LASH EXPAND'
BACK SLASH OR EXPAND

***************

Page 190

\[(12p1 0 0)' A P L '; (3p10) \tau 212 \times 30\]
A P L \ 3 6 0
Index
INDEX

absolute value function, 68
AB program, 68
ABS program, 68
ABSOLUTE VALUE program, 68
adding (lines to a program), 24
addition function, 2
and function, 48
and-reduction, 50
answers (to exercises), 218-238
APL, Preface
APL/360, About APL Systems, 187
APL Press, Preface
applications (of functions), 77-96, 97-118
arc-trigonometric functions, 173
ARCCOSH program, 173
ARCCOSINE program, 173
ARCSINE program, 173
ARCSINH program, 173
ARCTANGENT program, 173
ARCTRIG4 program, 173
AREA program, 20
arithmetic drill-and-practice, 102
arithmetic functions, 2, 164
array functions, 139-162
arrays, 119-138
assignment command, 4
ATTN (attention) key, 56, 84
AVERAGE program, 31
axis (of reversal), 140

BASEBALL program, 22
base (of a logarithm), 169
decimal number representation, 3

ceiling function, 71
CEILING program, 70
clocking, see catenation function
cumulative (solutions), 16, 32, 42, 49, 69, 70, 72, 90, 99, 113, 128, 142, 143, 150
changing (lines in a program), 21
character editing, see line editing
circular functions, 170
coefficients, 161
colon, 104
catenation function, 6, 9, 156
chip function, 71
CEILING program, 70
chaining, see catenation function
data, 17
data representation, 2
data structures, 133
deal function, 88
decimal number representation, 3
DECODE program, 185
definition error, 175

DEGREES program, 174
del, 20
deleting (lines in a program), 61
diagonal (of a matrix), 146
dimension (of an array), 133, 134
displaying (a program), 24
division function, 2
domain error, 66, 74, 178
DRILL program, 102
Drill-and-practice, 102, 135
drop function, 86, 132
dyadic scalar functions (summary), 200
dyadic functions, 25
dyadic program, 32, 60
e, 167
editing (a program), 21, 24, 57, 59-62, 104
summary, 212
element-by-element, see parallel processing
elements, 6, 120
elementing (characters), 104
ENGLISH program, 185
endless loop, 56, 84, 103
equals function, 11
ERASE command, 32, 175
error reports, 3, 4, 27, 31, 32, 66, 68, 81, 99, 110, 148, 175, 178
error reports (summary), 211
evaluation (of expressions), 30, 35-45
execution (of a program), 20
expansion function, 169
experiments, 14
explicit results (programs with), 62, 68

exponential function, 167
exponentiation, see power function
expressions, 17
execute function, 186
factorial function, 178
FAC program, 177
floor function, 71
FLOOR program, 70
format function, 186
functions, 17, 200-205
see also specific functions
ing function definition mode, 33
functions on arrays, 125
games, 79, 89, 96, 114
generalized reduction, 39
global (names), 33
go to (a line in a program), 56
grade-down function, 92
grade-up function, 91
greater-than function, 12
greater-than-or-equal function, 12

header editing, 60

Identity function, 164
incrementing, 58, 81
index-of function, 93
INDEX ERROR, 26, 27
index generator, see iota function
index origin, 95, 214
indexing, 26
initialization, 58, 81
inner product, 151
input, 98, 100
inserting (lines in a program), 57
interactive (programs), 97-118
iota function, 25
iteration, 56
Iverson, Kenneth E., Preface, 137

keyboard, Frontispiece, 100

lamination function, 158
Last-one-loses (game), 114
least squares regression, 162
LENGTH ERROR, 110, 148
less-than function, 12

INDEX 241
INDEX

less-than-or-equal function, 12
line editing, 104
line labels, 104, 114
lines (in a program), 20
literal data, 2, 100
local (names), 63, 81
logarithm functions, 168, 169
logical data, 48
logical functions, 48
LOL program, 114
loop, 56
LSORT program, 94
main program, 114
mathematical model, 116
matrices, 121, 133, 148
matrix, 121, 133, 148
matrix divide function, 161
matrix inverse function, 160
matrix product, 153
MAX program, 75
maximum function, 14
maximum-reduction, 40
MDRILL program, 137
membership function, 82
MEMBER program, 176
MIN program, 75
minimum function, 15
minimum reduction, 41
minus-reduction, 41, 130
miscellaneous functions, 163-191
mixed functions (summary), 203
mixed output, 108
mode, see command execution mode
program definition mode
model, 116
monadic scalar functions (summary), 202
monadic functions, 25
monadic program, 31
multiple assignments, 176
multiplication drill program, 102
multiplication function, 2
multiplication table, 150

names, 4
natural logarithm, 168
negation function, 164
negative number representation, 3
negative symbol, 3
Nim, 114
not function, 48
null vector, 25, 54, 64, 114, 134
numerical data, 2
nylaic (program), 33, 79

operators, see reduction, outer product,
inner product, scan
operators (summary), 206
or function, 48
or-reduction, 50
origin, see index origin

OUT, 101
outer product, 148
output, 2, 72, 150
overstrike (symbols), 10, 91, 92, 100, 123, 129, 131, 140-142, 156, 160, 168, 169
palindrome, 140
parallel processing, 8, 125
permutations, 90-92
pi-times function, 170
POW program, 55
POWER program, 60
power function, 65
POWOW program, 56
primitive functions, see functions
printing precision, 3, 150, 170, 214
problem-solving, 95, 138
program, 20
programming, 20
program definition mode, 19-34
quad input, 98
quad output, 72
quote-quad input, 100
quote marks, 2
radians, 171
RADIANS program, 175
RANDOM program, 135
random number generator, 78
RANDOM WORDS program, 81
rank (of an array), 134
ravel function, 124
reassignment, 5
reciprocal function, 166
recursion, 176
reduction, 30, 44, 129
relational functions, 11
renumbering (lines in a program), 57
replacing, see changing lines in a program
representation function, 182
residue function, 16
RES program, 69
restructuring function, 120
resuming (execution of a suspended program), 111
reversal function, 140
reviews, 17, 33, 44, 73, 96, 118, 138, 162, 191
rho, 26, 28, 120, 134
right-to-left (evaluation of expressions), 36, 41
ROLL program, 79
rotation function, 142
ROUND program, 72
rounding off (numbers), 72, 75
rows, 121
row-major order, 122
rules (for evaluating expressions), 33
RW program, 84
scalar, 2, 133
scan operator, 188
semicolon, 106
shared variable, see system variable
significant digits, 3, 150
signum function, 165
size, see rho
SORT program, 91
SPELLING program, 136
stacking, see lamination function
state indicator, 110, 112
statements (expressions in a program), 20
structure, see size
subprograms, 84, 114
subtraction function, 2
summary (of APL), 199-215
sum-reduction, 30, 39, 129
suspended program, 110, 111
syntax, 3
SYNTAX ERROR, 3, 31, 32, 68, 99
system command, 32, 175
summary, 213
system functions (summary), 215
system variable, 95, 150, 170
summary, 214
table, see matrix
take function, 85, 132
tangent function, 171
TANGENT program, 172
TANH program, 172
temper program, 116
time-reduction, 40, 130, 177
three-array, 133
trace command, 64
trace (a program), 64, 69, 75, 176
transpose function, 141, 145
TRI program, 87
TRIANGLE program, 23
TRIGO program, 172
TRIG4 program, 172
trigonometric functions, see circular functions
truth tables, 184
TRUTH program, 184

unconditional branching, 56

value, 4, 23
VALUE ERROR, 4, 81, 99, 101
variable, see names
vectors, 6, 133
VOWELCHECKER program, 83

242 INDEX
APL: An Introduction
Howard A. Peelle
This combination workbook/textbook offers a problem-solving approach to learning computer programming in APL. It is self-instructional, that is, you can teach yourself the APL language by using this book — with or without a computer.

Each chapter opens with an explanation of APL problem-solving tools, followed by numerous examples of APL expressions. The reader is then asked to solve selected exercises; these exercises can be done with or without a computer. Answers are provided in the appendix.

The book is written in an informal style, with handwritten annotations alongside examples of APL expressions that serve as supplementary explanation. Each chapter ends with a review, and summary tables of all expressions appear at the end of the text for quick reference.

Other Books of Interest...


and

ADVANCED BASIC: Applications and Problems
Both by James S. Coan
Two books that give you the complete picture of the BASIC language. One introduces the language; the other offers advanced techniques and applications. BASIC BASIC, #5106-9, paper, #5107-7, cloth, 288 pages; ADVANCED BASIC, #5855-1, paper, #5856-X, cloth, 192 pages

BASIC FROM THE GROUND UP
David E. Simon
An introduction to BASIC for the novice, covering all the features of BASIC as well as explaining the inside workings of the computer. Includes exercises and worked-out problems. #5117-4/Text, #5760-1/Trade, paper, 232 pages

PROGRAMMING PROVERBS
Henry F. Ledgard
Offers short rules and guidelines for writing more accurate, error-free programs. Contains standards for and programs in PL/1, ALGOL, BASIC, and several other languages. "This gem of practical guidance is much needed and long overdue." American Association for the Advancement of Science. #5522-6, paper, 144 pages