∧ ~ ⍱ ⍲ + − × ÷ ⋆ ? ∊ !

M[×ρM ← 1 1 ↓M-M[;1]] ← → ⍝

< > = ≠ ∇ ∆ ; :

≤ ≥ ∨ ⌹ ρ ↓ ⌊ ⍋

↑ ⍳ ⌽ . ○ ∘

⍟ ⌈

⍒

Learning APL: An Array Processing Language

( ) ⊤ [ ]

[1;1]÷D←1

James A. Mason

○

∘ ⌹ ρ

↑ ⍳ ⌽ ⍟ ⌈ ⍒ ! ↓ ⌊ ⍋

[×ρM;] ← D × M[×ρM ← 1 1 ↓M-M[;1]]°

LEARNING APL

AN ARRAY PROCESSING LANGUAGE

James A. Mason
York University

1817

HARPER & ROW, PUBLISHERS, New York
Cambridge, Philadelphia, San Francisco,
London, Mexico City, São Paulo, Singapore, Sydney

To my parents, Lucile and Keith

Sponsoring Editor: John Willig
Project Editor: David Nickol
Text Design: Betty L. Sokol
Cover Design: Betty L. Sokol
Production: Delia Tedoff
Compositor: Computer Data Systems, Inc.
Printer and Binder: R. R. Donnelley & Sons
VAX is a trademark of Digital Equipment Corporation.

Library of Congress Cataloging in Publication Data

Mason, James A.
Learning APL.

Bibliography: p.
Includes index.
1. APL (Computer program language) I. Title.
II. Title: Learning A.P.L.
QA76.73.A27M37 1985 001.64'24 84–25151
ISBN 0–06–044243–3

85 86 87 88 9 8 7 6 5 4 3 2 1

Preface

A typical programmer first learns computer programming in a language like BASIC, COBOL, FORTRAN, Pascal, or PL/I, in which data manipulations are performed in many small steps, each described by a separate statement. Usually even the simplest tasks to be performed in such a language require programs of more than ten statements, and most tasks require much longer programs.

APL is a very different programming language, since it permits the programmer to perform complicated data manipulations with extremely short programs and even with one-line expressions. The need for the programmer to write explicit loops, conditional statements, and assignment statements is minimized by the power of the functions that APL provides for manipulating arrays of data. Of all the common general-purpose programming languages, APL is the one in which application systems can be implemented fastest.

The main ideas of APL were developed by Kenneth Iverson and presented in his book *A Programming Language* (John Wiley & Sons, 1962), although the notation of APL has changed from Iverson's original version and many features have since been added to the language. The acronym APL comes from the title of Iverson's book. The subtitle of this book (*An Array Processing Language*) suggests a more specific interpretation of the acronym.

The goal of this book is to help the reader to acquire fluency in APL by learning to think in terms of arrays of data and by learning the large vocabulary of APL functions that operate on arrays. That learning is best accomplished by solving problems in APL. Hence, the most important parts of this book are the exercises. They have been chosen carefully to be short, but interesting and challenging, and to be cumulative in their effect. In order to be accessible to the widest possible readership, the exercises have been designed to require only high-school mathematics, for the most part, and to involve character data as much as numerical data. Many of the exercises suggest practical applications.

This book is intended primarily for readers who already have some experience with computer programming in a language other than APL. For such readers it is suitable either for self-study or for use in a one-semester course. The book may also be used in a course for novice programmers if it is supplemented with introductory material on the elements of computer usage and programming.

The book should be read as much as possible in sequence from beginning to end, although parts involving three-dimensional and higher-dimensional arrays may be skipped over on first reading, if desired. As many of the exercises as possible should be solved, and the answers tested on a computer. (Suggested answers to some of the exercises are given in an appendix.) Each exercise should be solved using only those features of APL that have been introduced up to that point in the book. In classroom use it is appropriate for some of the exercises to be used as examples in class and the rest to be assigned as homework.

The first six chapters cover most of the standard features of the APL language as well as some common extensions. The coverage is reasonably comprehensive but not exhaustive. In addition to this book it is desirable to have a reference manual for whatever implementation of APL is being used. That will provide details of features of APL that are beyond the scope of this book and that may differ from one version of APL to another. (Those include certain system commands, system variables, system functions, and file-accessing functions that are not part of standard APL.)

Chapter 7 illustrates a way of making APL expressions look like English sentences, a technique that has not been presented in other books on APL. It also provides a case study of how an application can be implemented quickly and easily as a workspace of many small, cooperating APL functions.

Having used and taught APL and other general-purpose programming languages for over 10 years, I have found APL about the most pleasant and easiest-to-use language of all. I am confident that after reading this book and working the exercises, the reader will have acquired fluency in APL and will have come to share my enthusiasm for this interesting and powerful programming language.

I would like to thank the following people for their help in producing this book (my sincere apologies to anyone I have overlooked): Lewis Baxter, Allan Cobb, Eric Drumm, Richard Levine, and Peter Roosen-Runge for their comments on earlier versions of the book; my students, especially George Stephen who contributed several exercises; my editors, John Willig and David Nickol, and sales representative Ellen Graca at Harper & Row, and the prepublication reviewers, for their helpful suggestions for improving the manuscript; my friends Beatrice and Marvin Mandelbaum, Jane and Lawrence Muller, and P. Rajagopal for their encouragement; Donald Solitar for his comments and for his invaluable help in producing the final copy of the manuscript; York University for providing computing facilities; and the staff of the Atkinson College Duplicating Centre for their careful and speedy work. Special thanks go to Thomas Plum for a conversation that inspired me to develop Chapter 7. Finally, I would like to thank Saundra, Eric, and Allan for their patience and support.

James A. Mason

Contents

Chapter 1

USING APL

1.1 The Unique Nature of APL

APL is a language of a very different sort from the more well-known programming languages COBOL, FORTRAN, BASIC, PL/I, and Pascal, and it is generally easier to use than any of them. The main power of APL resides in its functions for manipulating arrays of data. In many cases they eliminate the need for explicit step-by-step programming. In APL, short expressions can often be written to do what would require many statements in one of the more conventional programming languages.

Unlike most other popular languages, APL is a highly interactive language, designed to be used conversationally at a typewriter terminal or video display terminal. The APL user types one-line expressions and receives immediate responses from the computer. Data values are entered at the terminal and can be used immediately for computations or can be stored in a permanent *library* for later use. Programs, called *functions*, are also typed at the terminal and can be executed immediately or saved in the user's library. Because of its conversational nature and its conciseness, APL is a very pleasant language to use.

The next section of this chapter gives some short examples of sessions at an APL terminal. One thing you will notice from the example sessions is that APL uses an unusual set of characters. Except for its system commands, APL has no English keywords, and each primitive function in the language is represented by a single symbol (or in a few cases, two). The use of symbols rather than words makes APL expressions very concise, and understandable internationally. However, it does take some time and effort to get used to the APL symbols. So we will learn the meanings of the various symbols gradually. Appendix A shows where the APL characters are found on a normal APL keyboard.

1.2 Example APL Sessions

In the following pages examples of two APL sessions are presented. These sessions should be read not for full understanding, but to get an initial idea of how APL is used.

Lines typed by the person who was using the computer have been underlined for clarity. Also, some explanatory comments have been added.

After you have finished reading Chapter 1, you should try to recreate parts of the example sessions on your computer. Consult Appendix A for help in typing the special characters used in the examples.

Session 1

Sign on to the computer and entry to the APL system. (Exact details vary from one system to another.)

```
CLEAR WS                          —The initial workspace is clear (empty).
      HIGHS←3 ¯1 8 15 27 25 32 35 24 18 10 6     —assigning an array
                                                  of high
                                                  temperatures for the
                                                  months of a year, in
                                                  degrees Celsius
```

Note the raised negative sign (¯), not to be confused with the minus sign (−).

```
      HIGHS
3 ¯1 8 15 27 25 32 35 24 18 10 6 —displaying the array of
                                  high temperatures
      HIGHS[6 7 8]                —displaying the highs for June, July, and
                                  August
25 32 35
      LOWS←¯25 ¯22 ¯8 ¯3 ¯1 5 15 18 2 ¯4 ¯2 ¯10    —assigning an
                                                   array of low
                                                   temperatures
                                                   for the months
                                                   of the year
      ⌈/HIGHS                     —finding the highest of the highs
35
      ⌊/LOWS                      —finding the lowest of the lows
¯25
      (+/HIGHS)÷12                —computing the average of the highs
16.83333333
      ∧/HIGHS≥LOWS                —testing that each high is at least as large as
                                  the corresponding low
1                                 —1 means true.
      (HIGHS>25)/⍳12              —finding the months with high temperatures
 5 7 8                            greater than 25 degrees Celsius
```

```
      RANGE←HIGHS-LOWS  —finding the difference between the high and
                          low temperatures for each month
      RANGE             —displaying the temperature ranges for the 12
                          months
28 21 16 18 28 20 17 17 22 22 12 16
      (RANGE=⌈/RANGE)/⍳12 —finding the months with the maximum
                          temperature range
1 5
      ∇F←FARENHEIT C    —defining a new function
[1]  ⍝CONVERTS TEMPERATURES FROM CELSIUS TO FARENHEIT —a comment
[2]  F←32+C×1.8
[3]  ∇                  —end of function definition
      FARENHEIT HIGHS   —applying the function to the vector of highs
37.4 30.2 46.4 59 80.6 77 89.6 95 75.2 64.4 50 42.8
      )WSID TEMPS       —giving the workspace a name (TEMPS)
WAS CLEAR WS
      )SAVE             —saving the workspace on disk
13:24:16 18-FEB-84 3 BLKS TEMPS
      )LIB              —listing names of workspaces in the user's
                          disk library
GRADES
PRICES
TEMPS
      )OFF              —signing off
```

Sign off messages.

Session 2:

Sign on.

```
CLEAR WS
      )LIB              —Listing the names of workspaces in the
                          user's disk library
GRADES
PRICES
TEMPS
```

```
      )LOAD PRICES          —loading a workspace from the library
SAVED 19:56:50 17-FEB-84 2K

      )FNS                  —listing the names of user-defined functions
                            in the workspace
LIST

      ∇LIST[⎕]∇             —displaying the definition of the LIST
                            function
      ∇ QUANTITIES LIST PRICES;EXT

[1]     ' '
[2]     EXT←QUANTITIES×PRICES
[3]     '           UNIT  EXTENDED'
[4]     'QUANITITY PRICE     PRICE'
[5]     ' '
[6]     5 0 9 2 10 2 ⍕QUANTITIES,PRICES,[1.5]EXT
[7]     ' '
[8]     'TOTAL:          ', 10 2 ⍕+/EXT
      ∇

      )VARS                 —listing the names of variables in the
                            workspace
P       Q

      P                     —displaying the value of variable P
1.75 0.59 3.98 17.25 32.98 2.99 19.95

      Q                     —displaying the value of variable Q
3 8 12 2 9 0 1

      Q LIST P              —applying the LIST function to Q and P

          UNIT  EXTENDED
QUANTITY PRICE     PRICE

    3     1.75      5.25
    8     0.59      4.72
   12     3.98     47.76
    2    17.25     34.50
    9    32.98    296.82
    0     2.99      0.00
    1    19.95     19.95

TOTAL:            409.00

      ∇LIST[.1]             —inserting some new lines into the LIST
                            function
[0.1]       ⍝ COMPUTES EXTENDED PRICES AND TOTAL PRICE
[0.2]       ⍝ FROM GIVEN QUANTITIES AND UNIT PRICES
[0.3]       ∇
```

```
      ∇LIST[⎕]∇            —displaying the revised definition of LIST
     ∇  QUANTITIES LIST PRICES;EXT
[1]   ⍝ COMPUTES EXTENDED PRICES AND TOTAL PRICE
[2]   ⍝ FROM GIVEN QUANTITIES AND UNIT PRICES
[3]    ' '
[4]    EXT←QUANTITIES×PRICES
[5]    '          UNIT   EXTENDED'
[6]    'QUANTITY PRICE     PRICE'
[7]    ' '
[8]    5 0 9 2 10 2 ⍕QUANTITIES,PRICES,[1.5]EXT
[9]    ' '
[10]   'TOTAL:          '; 10 2 ⍕+/EXT
     ∇

      )ERASE P Q           —erasing variables from the workspace

      )VARS                —checking whether any variables remain

      )WSID                —displaying the name of the current active
                           workspace
PRICES

      )SAVE                —saving the workspace in the disk library
14:39:52 19-FEB-84 4 BLKS PRICES

      )LOAD TEMPS          —loading the workspace saved in Session 1
SAVED    13:24:16 18-FEB-84 2K

      )VARS                —listing names of variables in the workspace
HIGHS    LOWS    RANGE

      HIGHS                —displaying the value of HIGHS
3 ¯1 8 15 27 25 32 35 24 18 10 6

      LOWS                 —displaying the value of LOWS
¯25 ¯22 ¯8 ¯3 ¯1 5 15 18 2 ¯4 ¯2 ¯10

      MONTHS←⍳12           —creating an array of month numbers

      MONTHS               —displaying the value of variable MONTHS
1 2 3 4 5 6 7 8 9 10 11 12

      MONTHS[⍋HIGHS]       —displaying month numbers in order of
                           increasing high temperatures
2 1 12 3 11 4 10 9 6 5 7 8

      MONTHS[⍋LOWS]        —displaying month numbers in order of
                           increasing low temperatures
1 2 12 3 10 4 11 5 9 6 7 8

      )CLEAR               —clearing the active workspace
CLEAR WS
```

```
      ⍝ SOME SHORT EXAMPLES:    —a comment
      ⍝ POWERS OF TWO:
      ×\16⍴2
2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536
      ⍝ A MULTIPLICATION TABLE:
      TABLE←(⍳10)∘.×(⍳10)
      TABLE
  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
      ⍝ BIG LETTERS:
      BIGO←10 10⍴'O'          —creating big letter 'O' in a 10-by-11
                               character matrix
      BIGO[2+⍳6;2+⍳6]←' '
      BIGO[1 10;1 10]←' '
      BIGO←BIGO,' '
      BIGF←10 11⍴'FF',9⍴' '  —creating big letter 'F' in another 10-by-11
                               character matrix
      BIGF[1 2;⍳10]←'F'
      BIGF[6 7;⍳6]←'F'
      BIGO,BIGF,BIGF         —printing a word made from the big letters
 OOOOOOOO  FFFFFFFFFF FFFFFFFFFF
OOOOOOOOOO FFFFFFFFFF FFFFFFFFFF
OO      OO FF         FF
OO      OO FF         FF
OO      OO FF         FF
OO      OO FFFFFF     FFFFFF
OO      OO FFFFFF     FFFFFF
OO      OO FF         FF
OOOOOOOOOO FF         FF
 OOOOOOOO  FF         FF
      )OFF                   —signing off
```

Sign off messages.

1.3 System Commands

Before we get into the APL language itself, you must know a few things about the APL system environment: how to sign on and off from APL, and how to save and retrieve things in your private library for permanent storage of programs and data (which is usually on a disk storage device). Many of the system commands summarized in this section have already been illustrated in the sample terminal sessions.

A key element of the APL system is the *workspace.* When you sign on to an APL system you are given a clear workspace into which you can enter data and functions, and in which you can perform computations. The workspace you are using at any one time is called the *active workspace.* Whenever you wish you can save the entire active workspace, including the data and functions in it, in your library. The library can contain many workspaces, which remain stored when you sign off from the computer. At any time when you are signed on, it is possible to *load* a workspace from your library. The workspace loaded will then become the active workspace.

A typical APL session consists of the following sequence of steps:

1. Sign on to the computer and enter the APL system.

2. Use the initial clear workspace, or load a workspace from your library.

3. Enter data, enter function definitions, and do computations in the active workspace.

4. Repeat as often as desired:

   a. Save the active workspace in your library (if you want to keep it).

   b. Obtain a new clear workspace or load another workspace from your library.

   c. Enter data, enter function definitions, and perform computations in the active workspace.

5. Save the active workspace in your library (if desired).

6. Sign off from the computer.

To use APL you need to know the following system procedures and commands. System commands begin with a right parenthesis to distinguish them from names of variables or functions.

How To Sign On The procedure for signing on to APL varies from one computer to another. Generally the procedure is quite simple, but you will have to learn it from manuals for your particular computer and terminal.

How To Sign Off Signing off is easier than signing on but no less important. If you forget to sign off, and instead simply turn off your terminal, various undesirable things may happen: You may lose data, or you may incur extra usage charges if you are paying for computer usage. To terminate an APL session properly you should type the system command

```
        )OFF
```

In reply the computer will type an appropriate message, perhaps including some job statistics, such as the cost of the session and the balance remaining in your account.

How To Get a Clear Workspace When you sign on to APL the active workspace is initially clear, containing no data or functions. If at some other time you want to replace the active workspace by a clear workspace, type the system command

```
        )CLEAR
```

That command will destroy all user-defined functions and data currently in the active workspace.

How To Give a Name to the Active Workspace Workspaces stored in your library must have distinct names, called *workspace IDs*. You can give a workspace a (new) name when it is active by typing

```
        )WSID name
```

including the name which you have chosen, as in this example:

```
        )WSID PROJECT
```

In reply, the system will give the old name for the workspace, or will type *WAS CLEAR WS* if the workspace was unnamed. Different versions of APL have different rules regarding what are acceptable workspace names. Usually a