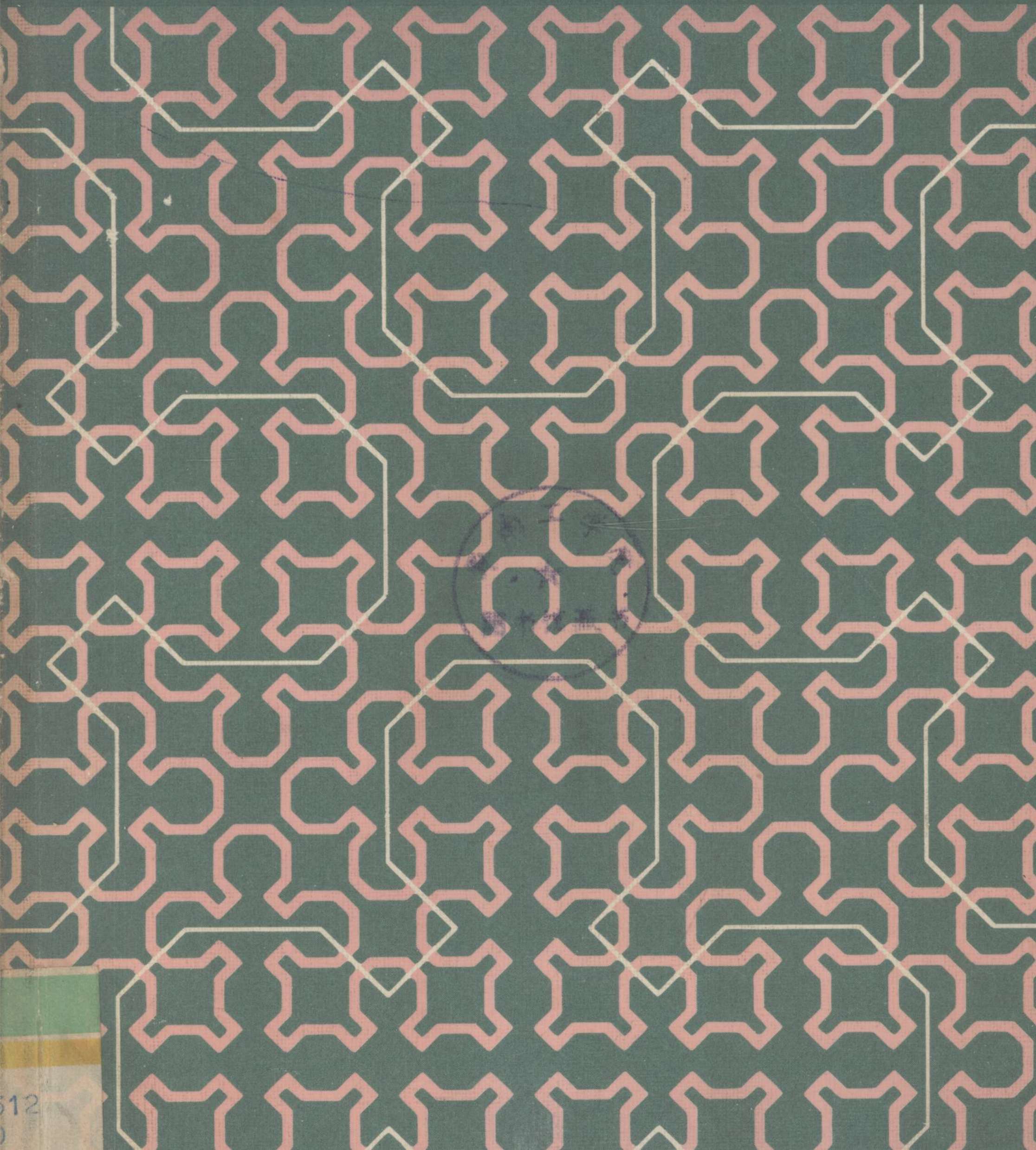
Cambridge Computer Science Texts · 17

An introduction to APL

S. POMMIER



TP312
P10

Cambridge Computer Science Texts · 17

An introduction to APL

S.POMMIER
Collective name for a team at the Ecole Nationale Supérieure des Mines comprising:
Jean-Jacques Girardot
Serge Guiboud-Ribaud
Bertrand Jullien
Francois Mireaux
Michel Nakache

Translated by Bronwen A. Rees

CAMBRIDGE UNIVERSITY PRESS
Cambridge
London New York New Rochelle
Melbourne Sydney

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge CB2 1RP
32 East 57th Street, New York, NY 10022, USA
296 Beaconsfield Parade, Middle Park, Melbourne 3206, Australia

Originally published in French as *Introduction à APL* by
Dundod informatique, Paris and © BORDAS, Paris, 1978
First published in English by Cambridge University Press 1983 as
An introduction to APL
English edition © Cambridge University Press 1983

Printed in Great Britain at the University Press, Cambridge

Library of Congress catalogue card number: 83-7374

British Library cataloguing in publication data

Pommier, S.
An introduction to APL – (Cambridge computer science texts; 17)

1. APL (Computer program language)
I. Title II. Rees, B. A.
III. Introduction à APL. English
001.64′24 QA76.72.A27

ISBN 0 521 24977 5 hard covers
ISBN 0 521 27109 6 paperback

DJ

PREFACE

APL originates from the work of K. E. Iverson, during his time as Professor at Harvard, on formalising algorithms. Its fundamental concepts are outlined in his book, *A Programming Language*, published in 1962. It was first used on a computer at IBM in the late 1960s.

The Department of Computer Science of the Ecole des Mines at Saint-Etienne developed several APL systems. In recent years new APL systems have been appearing and the use of the language is beginning to spread.

The present manual is designed to guide the first steps of the APL user. Consequently, only those aspects which we believe to be essential for a graded study in computer science have been detailed - the reader will not find an APL reference manual for a specific computer here. Numerous examples have been given to illustrate the text. They have been programmed on our system and the responses shown are those that actually occurred. We have tried our best to avoid particular cases, but where this has been impossible we have informed the reader in a footnote. In order to help the reader two appendices appear at the end of the text:

The current APL alphabet,
An APL mini-guide.

CONTENTS

1 INTRODUCING THE TERMINAL

1.1 From slide rule to APL

The last few years have witnessed the development of more and more sophisticated methods of calculation.

Just as the slide rule has become a museum piece, pocket calculators which could only carry out a few operations have been superseded by pocket micro-computers, capable of being programmed in a developed language. Nevertheless, these machines still have their limitations, particularly in their capacity for storing data. The moment one needs to carry out operations of any significant size, it becomes necessary to use a 'true computer' the size of which is decreasing from year to year if not from month to month. Which language should be used? Without doubt, because of its power and (debugging) capabilities, APL is one of the most useful. It is now available on a wide range of computers from the very large to the very small.

To work in APL then, it is necessary to have access to a computer capable of understanding the language, that is, capable of understanding or executing the calculations demanded of it by the language user. In this chapter we shall examine how the APL machine can be used. We shall call it indiscriminately the APL interpreter or system.

1.2 The terminal and the computer

In order to make use of the computer's ability to perform calculations rapidly, we obviously must be able to transmit the necessary instructions to it. For this a *terminal* (sometimes called a *console*) is used. A terminal consists of a keyboard, similar to that of a typewriter. Through this keyboard the user instructs the computer. In addition, part of the terminal is designed for the computer to communicate results (or other information) to the user, in response to instructions he has given. The technology of this part, the 'output device', varies according to the terminal model.

Generally, they are of two types: a visual display unit (like a television screen), or a line printer that prints on paper (like a typewriter). However, the answer supplied by the computer is independent of the type of terminal used.

The terminal is either linked to the computer directly of via a telephone line depending on where it is situated. The data typed in by the user on the terminal is sent to the computer which subsequently communicates the result.

1.3 Entering instructions

Each character entered on the keyboard is transmitted to the computer which then displays the character on the screen (or piece of paper). The computer must be told when a line (also called an *instruction*) has been completed and this is done by depressing the 'carriage return' key. This indicates that it is now (and only now) that the computer should execute the calculation demanded of it. The carriage (or cursor if using a screen†) then moves to the beginning of the next line. We shall see later how to correct typing errors.

Unfortunately, different keyboards are not all identical and the reader should try to familiarise himself with each of them. Figure 1 gives an example of one. On the majority of keyboards, certain keys represent up to four characters (see Figure 2). In APL only the characters situated on the left hand side are used. To enter the characters situated in the upper part of the left hand side the relevant

† It is only through linguistic misuse that we still tend to use the term 'carriage'. This can refer to either carriage or cursor depending on the type of terminal used.

Figure 1

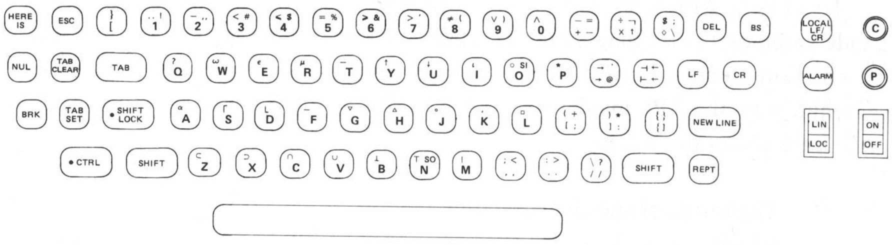
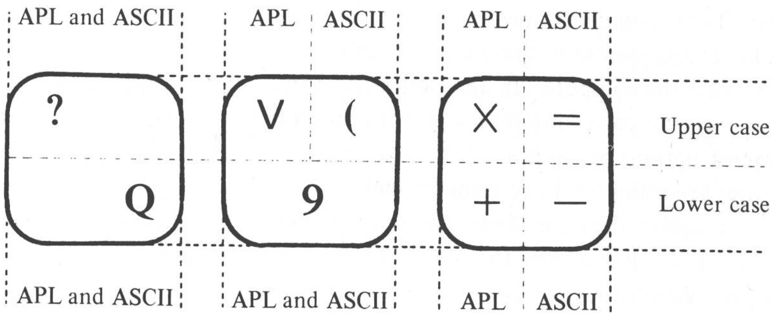


Figure 2



key and the shift key should be struck simultaneously. A special selector, APL/ASCII, makes the choice between characters on the right or left. The set of characters available in APL is the same whatever the terminal used:

```
0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZ ( ) [ ]
+−×÷*○⌈⌊|<≤=≠≥>∧∨~ αειρω?¨¯_↑→↓←⊃∩∪⊂⊥⊤ .,:;/\⎕'∘∆⍀
```

1.4 The computer's reply

When the calculation has been completed, the computer prints the result at the terminal and awaits further instruction on the next line, having shifted the carriage six spaces to the right.

An APL instruction often consists of no more than a few characters, but it can, on the other hand, take up more than one line of the terminal (in the physical sense). Equally, an instruction might simply consist of 'carriage return' to move to the following line.

1.5 APL expressions

A simple APL expression might consist of a single number or of two numbers separated by a function, thus:

```
      6
6


      3×5
15
```

Numbers are expressed in decimal form; if any part of the number is a fraction, it is separated from the integer part by a decimal point. The integer or the fractional part can be omitted if it is equal to zero. For example:

```
      .5+2.75
3.25


      1.7+2.3
4
```

Results supplied by the computer are printed following the same convention. In particular, the fractional part of a result will only be indicated if not equal to zero. But, for example, the terms 3, 3. and 3.00 are equivalent.

We have all used certain functions of the APL system from a very tender age. These are addition, subtraction, multiplication and division, represented by the keys +, −, ×, and ÷. These functions, used to carry out the majority of everyday numerical calculations, are called dyadic because two operands are necessary.

Thus:

```
      2×3
6
      1521+873
2394


      7865÷715
11
      605-872
¯267
```

In this example a new sign ¯ appears which indicates that the number is negative. Let us try using it:

```
      ¯1
¯1


      -1
¯1
```

The machine obstinately yields the result ¯1. Does this mean that the two keys — and ¯ have the same effect? In reality, no. In the first instance, — is the unary function (also called the *monadic* function) which, when applied to a number, gives its opposite value. In the second case, the expression is composed of the single constant, 'minus one'. The sign ¯ is a symbol used to represent negative numerical constants.

Also, this example shows us how a symbol such as — can have two distinct functions, depending on whether it is applied to a single number (or, more generally, operand) or to two. This is true for the majority of symbols used in APL. Let us try to discover the meanings of the other monadic functions corresponding to the keys +, ×, and ÷:

```
      +5
5
      +¯8
¯8
      ×5
1


      ×2
1
      ×¯3
¯1
      ×0
0
```

We can see that the × key, when used in its monadic form, gives the sign of its operand: 1 if positive, 0 if zero, and ¯1 if negative. Let us continue our investigation:

```
       ÷2
.5
       ÷3
.333333
       ÷¯1
¯1


       ÷0
DOMAIN ERROR
       ÷0
       ∧
```

This is rather a strange reaction from a machine that until now has proved to be quite amenable. It is, in fact, quite simply, an 'error message'. The machine is not capable of executing a division by zero. As the machine indicates, we have left the *domain* of the definition of the function used (in this case, the function of division).

Furthermore, the symbol ∧ shows precisely where the error lies, which is very useful when we have a long expression comprising several functions.

1.6 Exponential notation

Let us calculate:

```
       12300×27200
3.3456E8
```

In giving the answer, the machine adopts a neat notation: E8 means 'multiplied by 10 to the power of 8'. This representation is always used when the answer moves outside the domain in which the usual notation is clearer. Thus the computer will print 125 or 0.23, and not 1.25E2 or 2.3E¯1, but 1.3E9 or 3.65E¯11 and not 1300000000 or 0.0000000000365. However, in an expression, the terms 1E5 and 100000, for example, are strictly equivalent.

1.7 Error messages

Generally speaking, an error message indicates that the computer finds it impossible to execute a task properly. One of the more common reasons might be a mathematical impossibility:

```
       3÷0
DOMAIN ERROR
       3÷0
        ∧
```

Another cause might be the machine's limitations. For example:

```
        1.234E567
LIMIT ERROR
        1.234E567
        ∧
```

In fact, the internal specification for the representation of numbers means that they must lie within a certain interval, for example:

[¯1.701411E38 1.701411E38]

this interval varying from one computer to another.

Such an error, generally called a *'limit error'*, can also occur in any intermediate calculation:

```
        1E20×3.5E30
LIMIT ERROR
        1E20×3.5E30
            ∧
```

Other errors may result from the incorrect format for a number being used. For example:

```
        ¯  125
SYNTAX ERROR
        ¯  125
        ∧
        1 . 5E 4
SYNTAX ERROR
        1 . 5E 4
              ∧
```

In fact, spaces are not allowed between the negative sign and the number, or within the number itself or the exponent.

Another type of error occurs when an expression does not mean anything to the computer. For example:

```
        1+
SYNTAX ERROR
        1+
         ∧
```

1.8 Some other APL functions

(*a*) *Power and exponential*

These functions are symbolised by the key ∗ which in its monadic form represents the exponential, and, in its dyadic form, a power.

```
        3*2
9

        4*3
64
```

A power can also be a fraction or a negative number:

```
        2*.5
1.41421
        4*¯1.5
.125
```

As stated in its monadic form, the symbol represents the exponential:

```
        *1
2.71828


        *2.3
9.97419
```

(*b*) *Absolute value*
This function is represented by the sign |:

```
        |5
5

        |¯3
3
```

(*c*) *Rounding up or down*
Numbers can be rounded up or down by means of the 'ceiling' and 'floor' functions. Thus:

```
        ⌈3.14
4
        ⌊.5+2.63
3
```

The normal operation for rounding a number up or down can thus be written:

```
        ⌈¯.5+2.17
2
        ⌈¯.5+2.63
3
```

When the number is already an integer, the two functions give the same result.

```
        ⌈312
312

        ⌊312
312
```

When used in their dyadic form the maximum (⌈) or minimum (⌊) of two numbers can be calculated.

```
          23⌊17
17
```

1.9 The concept of variables

All the APL expressions that we have encountered so far have contained only numerical constants. However, it is possible in APL to retain a numerical value while designating a symbolic name to it, in the same way that in mathematics 'pi' or 'e' are used to denote the values 3.141592 .... and 2.71828 .... or 'X0' to denote a starting point in iteration, etc.

In APL the symbol ←, called the 'specification (or assignment) function', is used to create a name-value pair which we call a 'variable'.

```
          PI←3.141592
          X0←1
```

These operations result in the creation of two variables, called PI and X0 whose values are 3.141592 and 1. In an expression, these variables are referred to by their symbolic names.

```
          X0
1
```

```
          X0+2
3
```

```
          PI+X0
4.141592
```

When the APL interpreter comes across the name of a variable, it replaces it with the designated value of the variable.

```
          3×PI
9.424776
```

```
          3×3.141592
9.424776
```

It is, of course, possible to change the value of a variable, at any given moment; we say that we are assigning a new value to this variable:

```
        X0←2.5
        2×X0
5
```

The assigned value can be the result of a calculation:

```
        PI←PI+1
        PI
4.141592
```

In this example the expression PI + 1 was evaluated first, giving the result 4.141592, then this new value was assigned to PI.

The name (or *identifier*) of a variable consists of a string of letters (which may be underlined) and numbers, the first character being, of necessity, a letter. The maximum length of a name depends on the APL system used (for example 32 or 72 characters).

An error message occurs if an identifier that has not been specified is used:

```
        TOTO+3
VALUE ERROR
        TOTO+3
        ∧


        TOTO
VALUE ERROR
        TOTO
        ∧


        TOTO←2
        TOTO+3
5
```

'VALUE ERROR' means then, quite simply, that the computer cannot find the value associated with the name 'TOTO'.

1.10 How to use the APL system

At present, APL systems are available on a number of computers and in many different forms. There are microcomputers specially designed for APL which, like a pocket calculator, are ready to execute APL instructions as soon as they are switched on. On other machines APL is available in the form of a program called by a specific command. Finally, certain organisations offer an APL service as part of a time-sharing system, where the user identifies himself by entering in a user's sign-on number, and a password.

We shall not, however, describe the APL connection procedure since it varies dramatically from one system to another. For any given system you should refer to the manual supplied by the designer.

1.11 An example of a session

The following session took place on a T1600 computer, designed solely for APL, which operates a time-sharing system enabling several users to work simultaneously, each using his own terminal to communicate with the machine.

When a terminal is vacant it displays the following:

```
    APL/16\MINES   READY
RELEASE 3.0 DECEMBER 78 T030
        [redacted]
```

To use APL service, you must first of all identify yourself by giving a user's *sign-on number* followed by a *password.* Theoretically, the password is known only by the person who uses the sign-on number. This procedure is called *connection* (sign-on or log-in).

To connect up, a closing bracket, followed by the sign-on number, a colon and the password must be typed in. For example:

```
        )31415:SESAME
```

If the system recognises the sign-on number and the password, it gives an answer:

```
030)   83/03/21   10.08.15   SMITH



    APL/16\MINES   READY   64 K
```

Different APL systems can hide the sign-on number and password, either by not printing the characters typed in by the user, or by disguising them. This is what happened in our example.

This message gives the following information:

$$\underbrace{\text{030)}}_{\text{Terminal number}}\quad \underbrace{\text{83/03/21 10. 08. 15}}_{\text{Date and time of connection}}\quad \underbrace{\text{SMITH}}_{\text{User's name}}$$

The APL system is now ready for use.

When you no longer require the computer's services, you must indicate this with the message:

```
        )OFF
```

sometimes followed by a colon and a new password. The system then sends back the reply:

```
030   83/03/21   10.08.35   SMI
CONNECTED    0.01.00  TO DATE    0.01.00
CPU TIME     0.00.00  TO DATE    0.00.00
```

which contains the following information:

$$\underbrace{\text{030}}_{\substack{\text{Terminal}\\\text{number}}}\quad\underbrace{\text{83/01/05 10.08.35.}}_{\substack{\text{Date and time of}\\\text{disconnection}}}\quad\underbrace{\text{SMI}}_{\substack{\text{1st 3 letters}\\\text{of user's name}}}$$

$$\text{CONNECTED}\quad\underbrace{\text{0.01.00}}_{\substack{\text{Length of session}\\\text{in hours, minutes}\\\text{and seconds}}}\quad\text{TO DATE}\quad\underbrace{\text{0.01.00}}_{\substack{\text{Sum of the time of}\\\text{all the user's}\\\text{sessions}}}$$

$$\text{CPU TIME}\quad\underbrace{\text{0.00.00}}_{\substack{\text{Calculation time}\\\text{in hours, minutes}\\\text{and seconds}}}\quad\text{TO DATE}\quad\underbrace{\text{0.00.00}}_{\substack{\text{Sum of time for}\\\text{all of the user's}\\\text{calculations}}}$$

CPU stands for central processing unit.

This is what appeared on the user's terminal:

```
   APL/16\MINES  READY
RELEASE 3.0 DECEMBER 78 T030
      )[redacted]
030)  83/03/21   10.10.08   SMITH

   APL/16\MINES  READY  64 K

      1+2+3
6
      1+4÷3
2.33333
      E←*1
      E
2.71828
      *2.3
9.97419
      ∊
SYNTAX ERROR
      ∊
      ∧
```