APL—A TOOL OF THOUGHT

B.D. Joshi*

INTRODUCTION

APL stands for 'A Programming Language'. It was invented by Iverson as a notation for direct implementation and testing of ideas on a computer in an interactive environment.[1] Since then it has evolved into a general purpose programming language. Recently at least four versions of APL have become available for IBM PC.[2] This development, in conjunction with the popularity of IBM PC, has opened up a new dimension in personal computing, especially for mathematics and science oriented educational computing.

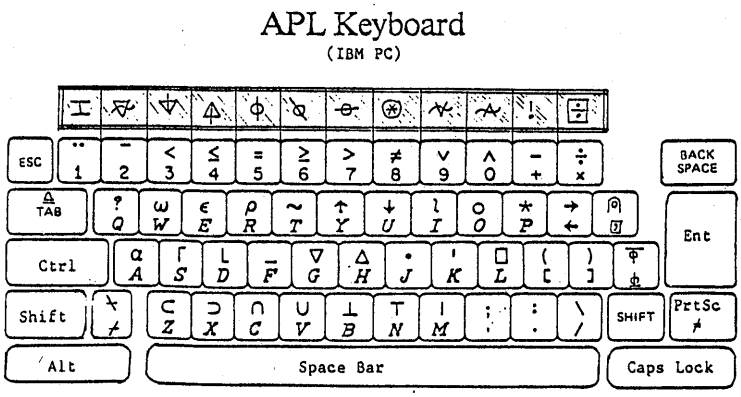
On a beautiful sunny day of May 1971 an aggressive IBM salesman left a shiny APL printing-terminal and an APL manual, for any one to play with, in Geneseo's then brand new Greene Science Building. The terminal was connected to an IBM main-frame computer at SUNY Binghamton. During that memorable summer I caught the "APL bug", and I am still thrilled by that summer's experience. For the next six years I did all of my scientific computing in APL. From 1978 to 1983 I taught a 3-credit hour APL programming course at Geneseo to a total of about 250 students. Last year I acquired IBM's APL for my IBM PC, and have been developing interactive user transparent problem solving packages in APL for use in our physical chemistry and related courses.

NATURE OF APL

Among general purpose programming languages currently avaàilable, APL stands in a class by itself.[3] There are three landmark papers that elucidate the nature of APL.[4-6] These along with five other basic articles dealing with APL have recently been reprinted as a single paperback book.[7] For beginning students of APL there are many textbooks and manuals available. The best known in this category is the book by Gilman and Rose.[8] In addition, there is a quarterly magazine devoted to a special-interest-group, consisting of persons interested in various aspects of APL, SIGAPL, has been organized within the Association for Computing Machinery.

Much of the terminology and symbolism used in APL is borrowed from the general domain of mathematics. It uses the precise concepts of scalars, vectors, matrices, and multi-dimensional objects for data structures. It also uses the well defined concepts of functions and operators for "data processing". In APL, *functions* modify the *data* and *operators* modify *functions*. Two types of *functions* are used in APL. A monadic function requires the data as its right argumant, and a dyadic function requires two pieces of data, presented as a left argument and a right argument. APL *operators* modify only the *dyadic functions*. There are at least 23 monadic and 53 dyadic intrinsic functions, often referred to as the primitive functions, available to APL programmers. In addition there are at least 5 general purpose operators used for modifying most of the primitive dyadic functions. Recursive programming is also supported in APL.

Single character symbols are used to represent APL *functions*. APL *operators* are represented by symbols containing one to three characters. The APL character set on IBM PC includes upper and lower case English Alphabets, some Greek Letters, and some special characters. The APL keyboard for IBM PC is shown in Fig. 1. It takes a while for new users of APL to get used to the APL character set.



Note: 1. Using 'Alt' in conjunction with the numeric keys in the top row produces characters shown in the shaded area, i. e., to generate a '⍵' hold the 'Alt' key down and press the '7' key.

2. Using 'Alt' in conjunction with an 'Alpha' key produces the corresponding lower case alphabet, i. e., to generate a 'q' hold the 'Alt' key down and press the 'Q' key.

Figure 1

APL is an array oriented language. Problems to be solved using the full power and capabilities of APL require a thought process for algorithm development not shared by traditional programming languages like FORTRAN, BASIC, and Pascal. I believe that this is one of the reasons why some people who have used FORTRAN, BASIC, and/or Pascal find if difficult to use APL. Since APL is an interpreter-based language, each line of the code is interpreted and executed separately. Thus, the strategies used for writing efficient APL programs are somewhat different from those used for writing programs in compiler based languages. Some APL programmers push this idea to an extreme by writing complete one-line programs which may be difficult for beginners to comprehend.

In APL no declaration of any type regarding the nature and size of variables is required. All input/output is generally format free. However, if needed, formatted output can be easily generated. The main burden on an APL programmer is that of developing creative, efficient, reliable, and robust algorithms for solving problems at hand.

Because of the tremendous flexibility provided by the available intrinsic functions of APL, usually no two APL programmers will create the same algorithm to solve a given problem. If no accompanying documentation for the algorithm exists, it may be difficult for an APL programmer to understand another person's program. Large APL programs are generally developed in modular form. The individual modules, called by the generic name, *functions* , are coded and tested separately. There exist strict rules for passing data from one module to the other. Data can also be shared by various modules in "global" form. All variables in APL are considered to be global unless they are localized in a given module. Variable-name conflicts among modules is easily avoided. This is accomplished by identifying all variables specific to a given module and localizing them in that module.

Science and mathematics oriented students seem to find it easy to learn and use APL. Students with no prior computing experience seem to learn APL easily, and generally faster, than those who have had "experience" with BASIC, FORTRAN, and/or other similar languages. Some formal background in college level algebra seems to be helpful in learning and mastering some of the primitive functions of APL. It takes a spirit of "adventure" and "discovery" and an easily available computer, i.e., a personal computer, to learn and practice APL. I have enjoyed learning, using, and teaching APL. The most remarkable thing about APL is that 14 years after first being exposed to it, I still enjoy learning, using, and teaching this unusual, and still evolving, language.

IBM's APL FOR IBM PC

To implement IBM's APL on an IBM PC you need (1) a graphic's monitor and card, (2) a 8087 chip - the numeric co-processor, and (3) a graphics printer. The APL software provided by the vendor includes the basic language interpreter, and two types of facilities called (a) application workspaces, and (b) auxiliary processors. There are five applications workspaces to help the user with

- printer management,
- full-screen editing of user defined functions,
- file management,
- asynchronous communications, and
- music samples.

There are six auxiliary processors to manipulate various devices of the PC:

- AP80 for graphics-printer control,
- AP100 for BIOS/DOS interrupt handling,
- AP205 for management of screen display,
- AP210 for DOS file management,
- AP232 for asynchronous communications, and
- AP440 for music generator.

The package comes with a 318 page reference manual.[10]

PROGRAMS DEVELOPED AT GENESEO

Two general purpose programs have so far been developed for use by Geneseo students on an IBM PC for data analysis and problem solving.[11] Each program contains about 16 different modules.

GSLINE: This program performs a linear regression of F(y) on G(x), given a set of (X, y) data and algebraic forms (user specifiable) of F(y) and G(x).[12]

NEWTON: This program solves <u>any</u> algebraic equation in one unknown,

$$F(a, b, c, \ldots ; y) = 0,$$

where a, b, c, etc are known parameters, and y is the variable of interest.

Figures 2 and 3 show two examples of the use of NEWTON in solving physical chemistry prob-

blems. The information displayed within square brackets is supplied by the user.

Figure 2 shows a solution of an equation,

$$y^3 + Ky^2 - y(k + cK) - kK = 0,$$

encountered in a more accurate calculation of the hydrogen ion concentration, y, in an aqueous solution of a weak acid. Here c is the molar concentration of a monoprotic acid with dissociation constant K, and k is the ionic product of water.

Figure 3 shows the use of NEWTON for solving the van der Waals equation for the molar volume of a gas.

```
                    Solve for y: F(a, b, c, ... ; y) = 0
                    DATA:  Type the following information:

F(a, b, c, ... ;y)=[ (y*3) + (KXyXy) - (yXk+cXK) + kXK                              ]
Derivative: dF/dy =[ (3XyXy) + (2XKXy) - k+cXK                                      ]
Initial guess: y0 =[ (KXc)*0.35                  ]  Convergence Threshold[ 1E-14 ]

              PARAMETERS                                  ITERATIONS
     ==============================           ==================================
     K=[ 1.00E-18 ]     [          ]           1    1.59882E-007
     k=[ 1.00E-14 ]     [          ]           2    1.22571E-007
     c=[ 0.100    ]     [          ]           3    1.05014E-007
       [          ]     [          ]           4    1.00338E-007
       [          ]     [          ]           5    1.00002E-007
     ==============================            6    1.00001E-007
                                               7    1.00000E-007
THE ROOT IS:    1.00000E-007
Converged in:    7 Iterations
F(y) at the root =   0.00000E000
|====================================|    ==================================
|Ent: To continue           Esc: To exit|
|Tab/Cursor Keys(←↑↓→): To locate field|
|     & type/change data in it          |
|====================================|
c-BJoshi/84
```

Figure 2

```
                    Solve for y: F(a, b, c, ... ; y) = 0
                    DATA:  Type the following information:

F(a, b, c, ... ;y)=[ ((P+a÷y*2)Xy-b) - RXT                                          ]
Derivative: dF/dy =[ P - (a÷y*2)X1-2Xb÷y                                            ]
Initial guess: y0 =[ RXT÷P                       ]  Convergence Threshold[  1E-8 ]

              PARAMETERS                                  ITERATIONS
     ==============================           ==================================
     P=[ 2X101325 ]     [          ]           1    1.10877E-002
     a=[ .364     ]     [          ]           2    1.10877E-002
     b=[ 4.27E-5  ]     [          ]           3    1.10877E-002
     R=[ 8.314    ]     [          ]
     T=[ 273.15   ]     [          ]
     ==============================

THE ROOT IS:    1.10877E-002
Converged in:    3 Iterations
F(y) at the root =   0.00000E000
|====================================|    ==================================
|Ent: To continue           Esc: To exit|
|Tab/Cursor Keys(←↑↓→): To locate field|
|     & type/change data in it          |
|====================================|
c-BJoshi/84
```

Figure 3

*Department of Chemistry
State University of Arts and Science
Geneseo, NY 14454